

Image Cover Sheet

CLASSIFICATION

UNCLASSIFIED

SYSTEM NUMBER

148743

**TITLE**

THE DAOR SEARCH AND RESCUE \ (SAR\) HELICOPTER TRANSIT MODEL

System Number:**Patron Number:****Requester:****Notes:****DSIS Use only:****Deliver to:** FF

DEPARTMENT OF NATIONAL DEFENCE
CANADA



OPERATIONAL RESEARCH AND ANALYSIS
DIRECTORATE OF AIR OPERATIONAL RESEARCH

DAOR RESEARCH NOTE 94/8

**THE DAOR SEARCH AND RESCUE (SAR)
HELICOPTER TRANSIT MODEL**

by

Y.S. Leung
G.L. Christopher

November 1994

OTTAWA, CANADA

 National
Defence

Défense
nationale

Operational Research and Analysis

CATEGORIES OF PUBLICATION

ORA Reports are the most authoritative and most carefully considered publications issued. They normally embody the results of major research activities or are significant works of lasting value or provide a comprehensive view on major defence research initiatives. ORA Reports are approved by DGOR and are subject to peer review.

ORA Project Reports record the analysis and results of studies conducted for specific sponsors. This category is the main vehicle to report completed research to the sponsors and may also describe a significant milestone in ongoing work. They are approved by DGOR and are subject to peer review. They are released initially to sponsors and may, with sponsor approval, be released to other agencies having an interest in the material.

Directorate Research Notes are issued by directorates. They are intended to outline, develop or document proposals, ideas, analysis or models which do not warrant more formal publication. They may record development work done in support of sponsored projects which could be applied elsewhere in the future. As such they help serve as the corporate scientific memory of the directorate.

DEPARTMENT OF NATIONAL DEFENCE
CANADA

OPERATIONAL RESEARCH AND ANALYSIS
DIRECTORATE OF AIR OPERATIONAL RESEARCH

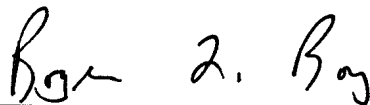
DAOR RESEARCH NOTE 94/8

**THE DAOR SEARCH AND RESCUE (SAR)
HELICOPTER TRANSIT MODEL**

by

Y.S. Leung
G.L. Christopher

Approved By:



Roger L. Roy, Director Air Operational Research

Directorate Research Notes are written to document material, which does not warrant or require more formal publication. The contents do not necessarily reflect the views of the Department of National Defence.

OTTAWA, CANADA

NOVEMBER 1994

ABSTRACT

The DAOR Search and Rescue (SAR) Helicopter Transit Model was developed to support the specification of minimum operating capabilities for a replacement SAR helicopter. For given set of helicopter characteristics and operating bases, the Model determines the optimal flight paths and transit times to travel to a distribution of available refuelling sites spread across the country. The Model was developed for the MULTICS computer system and used extensively to support the definition of the statement of requirements for a new SAR helicopter. After the statement of requirements for the SAR helicopter was approved, the Helicopter Transit Model was not used for some time. In that time the MULTICS system ceased operation.

In 1994, there was once again a requirement for the SAR Helicopter Transit Model. As the computer system for which the Model was designed was no longer operating, the Model had to be implemented on another computer system. The SAR Helicopter Transit Model was modified and implemented on a SUN Workstation. During the implementation process, the capabilities of the Model were enhanced.

This report documents the DAOR SAR Helicopter Transit Model and the work performed to return the Model to operational status. The report describes the procedures used by the Model to determine optimal flight paths and transit times. The report also provides a user guide to operate the Model.

RÉSUMÉ

Le modèle de Transit d'Hélicoptère de Recherche et Sauvetage (RES) a été développé pour établir les caractéristiques de performance pour un hélicoptère de remplacement en RES. Le modèle détermine les parcours optimaux et le temps de transit pour arriver aux différentes stations de ravitaillement à travers le pays à partir des bases d'opérations selon les caractéristiques de l'hélicoptère. Le modèle fut développé sur le système MULTICS et a été utilisé considérablement pour définir l'énoncé des besoins du nouvel hélicoptère SER. Depuis l'énoncé des besoins, le modèle n'a pas été utilisé. Pendant ce temps, le système MULTICS a terminé ses opérations.

En 1994, le modèle de Transit d'Hélicoptère RES a encore été requis. Vu que l'ordinateur sur lequel le modèle avait été développé avait cessé d'opérer, le modèle a été transporté sur un poste de travail SUN, et ses attributs modifiés et améliorés.

Ce rapport décrit le modèle de Transit d'Hélicoptère SER du DRO(A) et les modifications requises pour renouveler l'exploitation du modèle. Le rapport décrit les algorithmes utilisés par le modèle pour déterminer les parcours optimaux et les temps de transit. Un guide de l'utilisateur est aussi inclus.

TABLE OF CONTENTS

ABSTRACT/RÉSUMÉ	i
TABLE OF CONTENTS	ii
CHAPTER 1 - INTRODUCTION	1
BACKGROUND	1
OBJECTIVE	2
INITIAL IMPLEMENTATION ON SUN WORKSTATION	2
MODEL ENHANCEMENTS	3
IFR/VFR Consolidation	3
Output/Input Compatibility	3
Selection of Helicopter Bases	3
Display Enhancement	4
Multiple Helicopter Types	4
Model Efficiency	5
CHAPTER 2 - SAR HELICOPTER TRANSIT PROGRAM	6
GENERAL	6
INPUT SPECIFICATIONS	6
PARAMETER FILE	6
REFUELLING SITES DATA FILE	7
REFERENCE SITES (BASES) DATA FILE	8
TRANSIT TIME/PATH ALGORITHM	8
VFR AND IFR REQUIREMENTS	10
OUTPUT	11
CHAPTER 3 - MAP GENERATION PROGRAM	12
GENERAL	12
INPUT SPECIFICATIONS	12
MAP GENERATION	13
TRANSIT AND COVERAGE DATA	13
MAP PLOTTING PROCEDURES	13
OUTPUT	14
CHAPTER 4 - CONCLUSION	15
MODEL TESTING	15
TESTING THE MAPPING PROGRAM	15
TESTS ON THE HELICOPTER PROGRAM	16
CH-113/BELL 412SP COMPARISON	16
ANNEX A - HELICOPTER TRANSIT MODEL USER GUIDE	A-1
OPERATING THE TRANSIT PROGRAM	A-1
OPERATING THE MAPPING PROGRAM	A-5
VIEWING AND PRINTING THE SARMAP OUTPUT FILE	A-12
ANNEX B - PROGRAM LISTINGS	B-1
base.for	B-2
sarmap.for	B-13

SEARCH AND RESCUE (SAR) HELICOPTER TRANSIT MODEL

CHAPTER 1 - INTRODUCTION

BACKGROUND

1. The Search and Rescue (SAR) Helicopter Transit Model is an aid in establishing the performance characteristics for a replacement Search and Rescue (SAR) helicopter. Its task is to provide guidelines in choosing minimum helicopter capability to provide a rescue service for SAR distress accidents. Given the main operating bases, flying range, and speed of the helicopter type(s) available, the shortest transit time and path to reach different refuelling sites within the region of Canadian SAR interest are found. The resulting transit time and coverage using these bases are illustrated on a map of Canada and can be compared with the results of different configurations of helicopters and bases.
2. The model was originally developed and was working on the ORAE MULTICS computer in 1988. The implementation was written using Honeywell Fortran and was composed of two sub-models:
 - a. The first sub-model (Helicopter Transit Model) concentrated on finding the shortest time to reach each refuelling site. Two flying conditions under which the helicopter could operate when flying between SAR incident locations were considered:
 - (1) Instrument Flying Regulations (IFR), and
 - (2) Visual Flying Regulations (VFR).
 - b. The second sub-model (Map Generation Model) plotted map data, from files containing world map data, using a Calcomp plotter. A complete world map was generated for Centre-of-the-Earth and Mercator projections while a specified portion of the hemisphere was plotted for the Polar projection. It then took the results generated from the first model and overlaid them on the map.
3. The MULTICS computer was subsequently shut down and the SAR Helicopter Transit Model was not required until May 1994, when it was moved to a SUN Workstation and was enhanced to fulfil new modelling and analysis requirements.

- 2 -

OBJECTIVE

4. The aim was to return the SAR Helicopter Transit Model to operation so that it could be utilized to provide guidance in the selection of a helicopter to perform Search and Rescue in Canada with respect to:

- a. the minimum time of travel between locations (transit time),
- b. the maximum number of refuelling sites that can be reached (area coverage), and
- c. the minimum flying range and speed of the helicopter.

5. The purpose of the current activities was to implement the SAR Helicopter Transit Model on a SUN Workstation, modify the programs to meet the new requirements, and enhance the efficiency of the Model.

INITIAL IMPLEMENTATION ON SUN WORKSTATION

6. The model is currently operating on a SUN Workstation under the SUN OS environment. It is implemented using the SUN FORTRAN (Version 3.0) compiler.

7. The original Model was comprised of three programs: a helicopter transit program for visual flight (VFR) conditions, a helicopter transit program for instrument flight (IFR) conditions, and a map generation program. The first step taken in this project was to adapt the three programs to operate on the SUN Workstation. Minimal modifications were made to the helicopter transit programs, but several changes were required for the map generation program to function under the SUN environment.

8. The original map generation program utilized routines from a PLOT 10 Interactive Graphics Library (IGL) to produce the map diagrams. There is no equivalent PLOT 10 graphics library in the SUN FORTRAN compiler. There are, however, graphics routines in SUN FORTRAN which permit lines, polygons, and text to be displayed. To minimize modifications to the program and allow it to operate under the SUN environment, the PLOT 10 routines were translated into SUN FORTRAN graphics procedures. This approach maintained the program form as close to the original as possible.

9. The original map generation program produced output directly on an attached device. The program was changed to store the output in a postscript file. It is

- 3 -

assumed that a postscript output device is available for later printing. This change increased to portability of the model output.

MODEL ENHANCEMENTS

10. Several enhancements were made to the SAR Helicopter Transit Model:
 - a. IFR and VFR helicopter transit programs were consolidated into one program,
 - b. compatibility between helicopter transit program output and map generation program input were improved,
 - c. run time selection of helicopter bases was added,
 - d. display of the results was enhanced,
 - e. the Model was expanded to address multiple helicopter types, and
 - f. processing speed was improved.
11. IFR/VFR Consolidation. Except for parameters related to the different flying regulations, the two helicopter transit programs were performing the same basic operations to determine the shortest transit time and path. Therefore, it was logical and desirable to consolidate the two programs into one. An extra input "flag" was added to indicate the type of flight condition to be used. Separate procedures to handle the different flight rules were added. Finally program variable sizes were increased to accommodate the largest input data, irrespective of the flight rules to be used.
12. Output/Input Compatibility. Previously the output from the helicopter transit programs required modification before being used as input for the map generation program which was modified to run on the SUN Workstation. To eliminate this unnecessary action, the mapping program was further modified to accept the direct output from the helicopter transit program.
13. Selection of Helicopter Bases. In the original helicopter transit programs, the main operating bases for the helicopters being modelled were specified within the code of the program. Any change to the operating bases required the program to be re-compiled. While this might not be a problem if the bases remain static, it did limit the flexibility and ease of use of the Model.

- 4 -

14. In the original programs the operating bases were identified by a reference number which permitted the base to be related to a refuelling location which had associated latitude and longitude position values. The program then used these reference numbers to derive the helicopter transit paths.

15. The helicopter transit program was modified to read the names of the operating bases from a separate data file when the program is executed. The program then compares the base names to the list of refuelling locations to determine the refuelling point to which the base is related. The bases are then referred to in the transit program as they were in the original program. This procedure minimized the number of modifications that had to be made to the original program.

16. Display Enhancement. Several variations for plotting the map and helicopter transit routes were tried to obtain the best reasonable display of the results on a colour monitor and black and white postscript printer. Several different text fonts and sizes, and line textures and thicknesses were tested until a satisfactory result was obtained where the helicopter transit routes and times were clear and easily discernable from a map diagram both on monitor and paper form.

17. Multiple Helicopter Types. In the original model, one helicopter type was used. The flying speed and range were read from a parameter file or from interactive keyboard input. It was decided that being able to model more than one helicopter type in the simulation and find the optimal time and path to get to each refuelling site given the bases and characteristics of the helicopters would greatly increase the flexibility and utility of the Model. The helicopter transit program was adapted in the following ways:

- a. a maximum of 5 helicopter types can be used in one simulation,
- b. a list of bases is associated with each helicopter type indicating which type of helicopter is available at the base,
- c. helicopter characteristics (speed, range, number of bases used, and the names of the bases) are either read in from a data file or from interactive keyboard input,
- d. a many to one relationship between the refuelling sites and the helicopter types is established, i.e. there can only be one helicopter type serving a refuelling point, but different refuelling

- 5 -

points can be used by the same helicopter type,
and

- e. the helicopter type serving a particular refuelling site will depend on the base where the minimum transit time path originated and the helicopter type associated with the base.

18. Model Efficiency. A cursory review of the helicopter transit program was conducted to determine if any minor modifications could be undertaken to improve the processing efficiency. It was noted that the transit time between two sites depends on the speed of the helicopter and the distance between the sites. For each helicopter type, the transit time between two sites stays the same. Due to the nature of the algorithm used by the program to determine the minimum transit time path to a site, the processing time was reduced by grouping the bases by the helicopter types being modelled.

- 6 -

CHAPTER 2 - SAR HELICOPTER TRANSIT PROGRAM

GENERAL

19. This program determines the shortest transit time path to refuelling sites from different bases. Several factors determine the shortest time getting to a site:

- a. the specified bases from which the helicopters operate,
- b. the helicopter type that is available at the bases, and
- c. the flying regulation (IFR or VFR) being used in the simulation.

20. The reachable sites with the shortest possible transit time and path as well as the identification of the unreachable sites are stored in an output file.

INPUT SPECIFICATIONS

21. There are three groups of information needed to operate the transit model :

- a. refuelling sites,
- b. bases (referred to as reference sites in the program), and
- c. helicopter characteristics.

22. The first two types of information are contained in data files. If either of the files does not exist or contains errors, the program is aborted. These two files will be identified as the Refuelling Sites Data File and the Reference Sites Data File hereafter in this report.

23. The file names of these two data files are specified from either keyboard input or from a "parameter" file, if it is provided. Data representing the helicopter characteristics are provided in the parameter file, if it exists, or entered through interactive keyboard input.

PARAMETER FILE

24. The parameter file is optional. The default is interactive keyboard input,

- 7 -

which will be the case when the file name given for the parameter file is null. Information is required in the following sequence:

- a. file name of Refuelling Sites Data File,
 - b. IFR regulations applicable (.TRUE for IFR, .FALSE. for VFR),
 - c. file name of output file,
 - d. file name of Reference Sites Data File,
 - e. helicopter range,
 - f. helicopter speed,
 - g. number of bases (reference points) with this helicopter type, and
 - h. a list of names of the bases, corresponding to the refuelling sites in "a"; one name per line.
25. Steps "e" to "h" relate to helicopter information and are repeated until the end of input is detected. One complete loop defines a helicopter type, any incomplete loop will be treated as an error and the program will be aborted. Currently, the program accepts the specification of up to 10 different helicopter types.
26. An example of a parameter input file is provided in Annex A.

REFUELLING SITES DATA FILE

27. This file is compulsory. If a null filename is given, default files will be used (IFRSites.dat for IFR and VFRSites.dat for VFR simulation). The default data files are the ones specified for the original helicopter transit program which operated on the MULTICS computer system. An example of a Refuelling Sites Data File is provided in Annex A.
28. The Refuelling Sites Data File contains the name and the location (latitude and longitude in degrees and minutes) of the refuelling sites. There are more than 200 sites in each default file. Each line in the data file contains the data for one refuelling site. The data must be in a specific format (see Annex A) in order to be recognized.

- 8 -

29. The refuelling sites are given indexes according to the order in which they are read (i.e. the position in the file). These indexes are used to identify the refuelling sites and the bases in the program.

REFERENCE SITES (BASES) DATA FILE

30. This file is also compulsory. If a null filename is given, default file REFPT.DAT is used. The Reference Sites Data file contains information for the bases of the helicopters that will be modelled in the simulation. The default file contains information for five major bases: TRENTON, COMOX, GANDER, GREENWOOD and EDMONTON. A copy of a reference sites data file is provided in Annex A. Currently the program accepts up to 10 different helicopter bases.

31. The first data line of the file specifies the number of reference sites (bases) that will be used in the simulation. Each subsequent line of the file identifies a name for a reference site, base, which corresponds to the name of a refuelling site in the Refuelling Sites Data File. The program then compares the names of the reference sites and the refuelling sites to identify the matches and associates the appropriate location data with the reference sites. The location data for the associated refuelling sites will be used in any calculations involving the reference sites.

32. More reference sites than are being used can be present in the data file. Only the first sites specified by the number in the first line of the file will be used in the simulation. The extra sites will simply be ignored.

33. If a match between a reference site and a refuelling site cannot be established, the program is halted and the user is given a message indicating that there is a problem with the reference site input data. The user may then correct the problem and restart the program.

TRANSIT TIME/PATH ALGORITHM

34. The transit time to get to a site depends on several factors:

- a. location of site,
- b. reference sites (bases), and
- c. the flying range and speed of the helicopter types modelled.

35. The basic algorithm is comprised of two major steps. The first step is to find the transit time between every pair of sites which are not further apart than the range of the helicopter taking into account flight condition limitations. In the second

- 9 -

step, the reachable sites and shortest time path getting to them are found by simple comparison, with each path originating from one of the given bases.

36. When only one helicopter type is involved the algorithm is processed only once to produce the output results which identify the optimal transit time and path for the helicopters to travel from the operating bases to all the reachable refuelling sites. First the distance between every pair of refuelling sites or bases is calculated¹. If the distance is within the range of the helicopter, taking into account restrictions imposed by the flying regulations being used, a transit time is calculated from the helicopter's associated speed and assigned to the pair. For each site, all the reachable sites are recorded in ascending order of transit time.

37. Next a depth first search is performed, originating from each of the bases, to identify the shortest time path to each of the reachable refuelling sites. First, all the sites are assigned a zero transit time, used as key to identify unreachable sites. Then for each base, each reachable site is examined to determine that it is not faster to reach the site from another base. This yields all the sites that will be associated with the base. Then, for each of these sites, their reachable sites are examined to determine the shortest path to arrive at the reachable site. This process continues until the minimum travel time to every reachable site has been determined.

38. Every reachable site is given an associated minimum transit time and a parent site indicating where the helicopter would depart from to reach the site. Unreachable sites are identified by the lack of a parent site and an associated zero transit time marker.

39. When more than one helicopter type is being modelled, each helicopter type is evaluated independently and a comparison of optimal transit times is performed at the end. Each helicopter type is examined as if it were the only helicopter type involved. The optimal transit time and path to all the refuelling sites are determined and recorded for the helicopter type following the procedure defined above. After the results for each helicopter type have been determined, each refuelling site is examined and the minimum transit time and path are determined and recorded as the optimal transit time for the collection of helicopter types.

-
1. The distance (in nautical miles) between two sites is determined from:

$$D = \text{ARCOS}(\text{COS}(B) * \text{COS}(C) + \text{SIN}(B) * \text{SIN}(C) * \text{COS}(A)) * 10800 / \pi$$

where: A = Longitude of 2nd site - Longitude of 1st site (in radians)
 B = $\pi/2$ - Latitude of 1st site (in radians)
 C = $\pi/2$ - Latitude of 2nd site (in radians)

VFR AND IFR REQUIREMENTS

40. Two different sets of flight rules are available in the Model to assess the transit distance and speed requirements for SAR helicopters : Visual Flying Regulations (VFR) and Instrument Flying Regulations (IFR). These are the regulations under which the helicopter must operate when flying to and from SAR incident locations and limit the range and overall transit speed of the helicopter.

41. The refuelling sites which may be used during VFR conditions require only the availability of fuel. In the SAR Helicopter Transit Model, the transit time and coverage is determined by the flying range and speed of the helicopter type. The only requirement that must be satisfied in order to transit to a refuelling site is that the distance to the site from the point of departure must be less than or equal to helicopter flying range. In VFR conditions, the maximum helicopter transit distance is not influenced by the requirement for the helicopter to reserve fuel to be able to divert to an alternate refuelling site. All refuelling sites possessing jet fuel are available to the helicopter.

42. While transiting under VFR rules, a 30 minute delay is imposed on the helicopter at each refuelling stop. The delay is used to account for the time normally taken by the helicopter crew to refuel the aircraft.

43. Flying under IFR conditions, the helicopter must carry sufficient fuel to fly to the intended destination and in the event of bad weather, fly to an alternate. IFR bases must be equipped with landing approach navigational aids. The IFR-qualified refuelling sites are a sub-set of the VFR sites. The possible destinations for the helicopter are therefore limited by the helicopter range minus the distance to the nearest alternate. In the Model, a minimum of a 100 NM fuel reserve from the destination to an alternate is required to account for the geographic size of weather systems. Under IFR conditions, an approach time of 15 minutes plus 30 minutes to refuel and 15 minutes for a weather brief and takeoff, one hour, is added to the transit time between each pair of refuelling sites.

44. The requirements for refuelling sites to qualify as being reachable (a path exists) with helicopter operating under IFR conditions could be summarized as follows:

- a. the refuelling sites must be within a circle centred on the current position of the helicopter and having a radius equal to the flying range of helicopter,
- b. refuelling sites must be at least 15% of the helicopter range away from the current position, or else they will not be considered,

- 11 -

- c. upon arrival at a potential refuelling stop, the helicopter must have sufficient fuel remaining to reach an alternate refuelling site or travel 100 NM, whichever is greater.

OUTPUT

45. Results from the transit model are stored in an output file with a name specified in the parameter file. If a null file name is given, output will be defaulted to FLIGHT.IFR or FLIGHT.VFR for IFR and VFR simulation respectively. An example of a transit program output file is shown in Annex A.
46. The resulting output file contains information on the helicopters modelled, the shortest transit time and path to each reachable site, identifies the unreachable sites, and indicates how the sites are connected to produce the transit paths. The first line of the output file indicates the number of helicopter types modelled. For each helicopter type, three lines follow specifying the speed, time per refuelling stop, and the range of the helicopter, respectively. After the helicopter information, the next line identifies the number of reachable refuelling sites. This is followed by one line for each reachable site identifying the location of the site and the time (in hours) required to reach the site. Next there is a line showing the number of sites which are unreachable by the helicopters. This is followed by a line for each site, identifying the position of the unreachable site. Finally there is a series of lines in the output file indicating: the number of sites connected in the parent-child relationship (first line), and the position of each of the sites involved (following lines).
47. When one helicopter type is modelled, every reachable refuelling site has exactly one parent site which leads to it. That is, the refuelling site in question would be reached by the helicopter flying from the parent site. When more than one helicopter type is modelled, a reachable refuelling site may have more than one parent. There will be one parent which is used to provide the minimum transit time to the site. However, the refuelling site may be required by other helicopter types to reach more distant refuelling sites. These other helicopter types may arrive at the site from other different refuelling sites, and these other sites will also become parents to the site although they do not produce the minimum transit path to the refuelling site in question. For this reason, the last series of entries in the output file indicates the number of sites involved in the parent-child relationship first, followed by the positions of the sites.

CHAPTER 3 - MAP GENERATION PROGRAM

GENERAL

48. The original program plotted a map from files containing map data using a Calcomp plotter. The current program produces a map as output which is stored in a postscript file. Useful information can be overlaid on the map. In the SAR Helicopter Transit Model, it is only regions within, and adjacent to Canada and the results generated from the Helicopter Transit program that are of interest .

49. A complete map of Canada is contained in a data file named "g1.src". The data in the file represent boundaries, such as coastlines and lakes, which are plotted to generate the map. The location of the reachable refuelling sites with their shortest possible transit time and path as well as the location of the unreachable sites in the output file generated from the Helicopter Transit program are overlaid on the map.

INPUT SPECIFICATIONS

50. Two groups of information are needed by the Map Generation program: the map data, and the helicopter transit results.

51. As with the Helicopter Transit program, a parameter file can be used to specify the options to use to display the results. Again, if this file does not exist, interactive keyboard input is expected. An example of a map program parameter file is contained in Annex A. The parameters needed by the map program include:

- a. whether a new map is needed to be plotted,
- b. the file name(s) of information to be overlaid on the map,
- c. boundary of the map display region,
- d. projection type (lambert or mercator),
- e. resolution (high or low),
- f. the proportion of the page to be used to display the map, as a percentage, and
- g. titles for the figure.

52. The first parameter specifies whether a new map is to be generated or data are to be added to an existing map. If a new map is not to be generated, then the

- 13 -

program expects a postscript output file to be present. The second parameter identifies files whose data are to be overlaid on the map. One of the files will be the helicopter transit program results. The third parameter entry designates two latitude values and two longitude values to be used as boundaries for the map display. If data positions are outside of the region specified by the boundaries, they will not be displayed. Lambert or mercator projections can be selected. The resolution level dictates the amount of detail on the map. The size of the map is specified as the percentage of the page to be used. If 50 percent is specified the resulting map will occupy one half of the page. Finally, a two-line title can be specified for the map figure. If a title is not specified, one will be created from the name of the helicopter transit program output file.

MAP GENERATION

53. Using the recommended procedure (Annex A), a map is generated from a data file labelled "g1.src" which contains points specified by location (latitude and longitude) and resolution level (from 1 to 5). Depending on the resolution level chosen (high or low), only a portion of points are selected for display. Points with resolution level one are of higher resolution, providing more detail to the map, than points with resolution five. The selected points are then joined by line segments to form the map.

TRANSIT AND COVERAGE DATA

54. The helicopter transit information required by the map program are the transit times and paths (the shortest time path) generated from the helicopter transit program described in Chapter 2. The output file generated from helicopter transit program can be taken directly as input in the map program.

55. The file name of the input file containing the helicopter transit data is specified by the user. It should be noted that the file name will be used in labelling the map, if a title is not otherwise specified. The name of the data file should contain meaningful information to indicate the source of the data. For example, a file labelled "CH-113.IFR" which contains data generated for helicopter type CH-113 under IFR flying conditions would produce a map title: "CH-113 IFR TRANSIT TIME AND COVERAGE".

MAP PLOTTING PROCEDURES

56. The procedures used to generate the final map display can be separated into two classes: the programmer-written functions and the library functions. The programmer-written functions are graphics functions specific to the map generation program and utilize the graphics library functions to do the lower level graphics operations such as line drawing and point plotting. It is the higher level

- 14 -

programmer-written functions which are of interest and will be described.

57. The original program used graphics routines from the PLOT 10 IGL library to display the map and helicopter transit results. In the current program, procedures were written in FORTRAN to duplicate the functions of the PLOT 10 routines. This effort produced a local library of graphics routines to replace the PLOT 10 routines. This minimized the changes that had to be made to the original program.

58. The map grid is defined at the beginning of the procedure. It is the clipping region for the map content and is specified by the latitude and longitude boundaries and the resolution level to be used. The grid is a cartesian coordinate system defining the map display space and used to plot map features.

59. The map diagram is defined by line segments formed from connecting qualified points (determined by resolution of the map) read from the map data file. The points' latitude and longitude coordinates are converted to the screen coordinate grid for the selected projection type before the graphics routines are called.

60. The map is overlaid with information produced from the Helicopter Transit program. Three types of information are taken from the Helicopter Transit program output file and displayed on the map:

- a. a title is specified by the user or is derived from the output file name,
- b. refuelling sites that are reachable by helicopter from one of the bases are recorded on the map by placing the time to reach the site next to its position and connecting the paths to the sites by line segments, and
- c. displaying an asterisk on the map at the locations of the unreachable refuelling sites.

OUTPUT

61. The results of the map generation program are stored in the form of a postscript file. The results will be stored in a default file labelled "sarmap.out". The output file will be over-written each time the map program is executed. To secure the map results for long-term holding, the output file should be renamed before the map program is used again.

CHAPTER 4 - CONCLUSION

MODEL TESTING

62. The model has been tested on different conditions by varying the parameters passed to the two sub-models: the Helicopter Transit program and the Map Generation program. Tests on the mapping program were performed in the early development stage so that meaningful results could be represented in final output form. Major emphasis has been put on test results generated using different bases and helicopter types in the transit program, and reviewing the results using the map program.

TESTING THE MAPPING PROGRAM

63. The map program test consisted of two stages. The goal was to obtain the best possible graphical representation of the helicopter transit time and coverage results.

64. The first stage of the testing involved obtaining an optimal display of a map of Canada. This stage was performed before the transit program had begun to be modified. An attempt was made to find the best clipping grid that could plot a map covering as much area as possible and yet could still represent information clearly. The parameters used in the map program test included:

- a. the clipping region boundaries (latitudes 83 and 40 North, longitudes 150 and 50 West),
- b. the projection type (the mercator projection was found to be best),
- c. the resolution (the low resolution was decided upon as it provides adequate detail with the best display speed), and
- d. the map size (100 percent is always used as any size less than a full page does not permit the helicopter results to be discernable).

65. The second stage involved displaying the helicopter transit results over the map image. In this stage, the best computer monitor colours were sought to present the map and display the transit information on it such that locations of refuelling sites and the transit paths could be clearly seen. A compromise was sought between the best colours to display information on the computer monitor and the resulting

- 16 -

line shades produced on a monochrome printer. The thinnest stroke in a green colour was selected for plotting the outline of the map and produces a light shade when printed. A normal solid stroke, in red, is used for displaying the helicopter transit path which results in a dark solid line on the printer. The helicopter transit time to reachable refuelling sites is displayed in blue and prints as easily discernable bold characters.

TESTS ON THE HELICOPTER PROGRAM

66. Four bases were used in the tests, with different helicopter types and flying regulations being modelled. As the helicopter program was already operational and validated, no testing of the search algorithm for the optimum path was necessary. Testing during this phase of the project concentrated on validating the modifications made to the program to allow different helicopter types to be modelled at the same time. Simulation runs were made using the combinations of bases and helicopter types shown in Table I. Separate runs were made for VFR and IFR flight conditions as part of the test.

TABLE I : HELICOPTERS AND BASES USED IN MODEL TESTS

<u>Base</u>	<u>Helicopter Type</u>
Comox	CH-113 Labrador
Trenton	Bell 412SP Griffon
Greenwood	CH-113 Labrador
Gander	CH-113 Labrador

67. Results were generated for each base, with associated helicopter type, individually and for all the bases together for each flight regulation. Examining each of the individual base results and noting for each reachable refuelling site the best transit time would identify the result that should be present in the combined simulation run. Testing was completed when the combined results duplicated the optimal results from the individual base runs.

CH-113/BELL 412SP COMPARISON

68. The utility of the SAR Helicopter Transit Model was proven in a recent application to compare the performance of two helicopter types for the SAR role in

- 17 -

central Canada. The Canadian Land Force has procured a number of Bell 412SP helicopters to fill their requirement for utility aviation transport. The Directorate of Aerospace Requirements Fighters and Transport (DARFT), involved in the SAR Helicopter Replacement Programme, was interested in investigating the suitability of using the Bell 412SP helicopter for Search and Rescue tasks in the central region of Canada. A comparison of the SAR performance of the Bell 412SP with the current SAR helicopter, CH-113 Labrador, was requested.

69. Among the various criteria that were evaluated in the comparison, the area coverage and nominal transit time were to be examined using the SAR Helicopter Transit Model. The helicopter characteristics relevant to the area coverage and transit time evaluation are shown in Table II.

TABLE II : CH-113/BELL 412SP CHARACTERISTICS

	<u>CH-113</u>	<u>BELL 412SP</u>
Speed	108 kts	118 kts
Range	530 nm	310 nm
Base	Trenton	Trenton

70. It is clear from Table II that there is a minor difference in the cruising speed of the helicopters and a large difference in the range of the two helicopters. However, it is not possible to state whether these differences are significant in the performance of the SAR role until the helicopters are modelled to determine the area coverage they can provide. The helicopters' flight coverage in VFR and IFR conditions, using Trenton as the sole operating base, were simulated using the SAR Helicopter Transit Model.

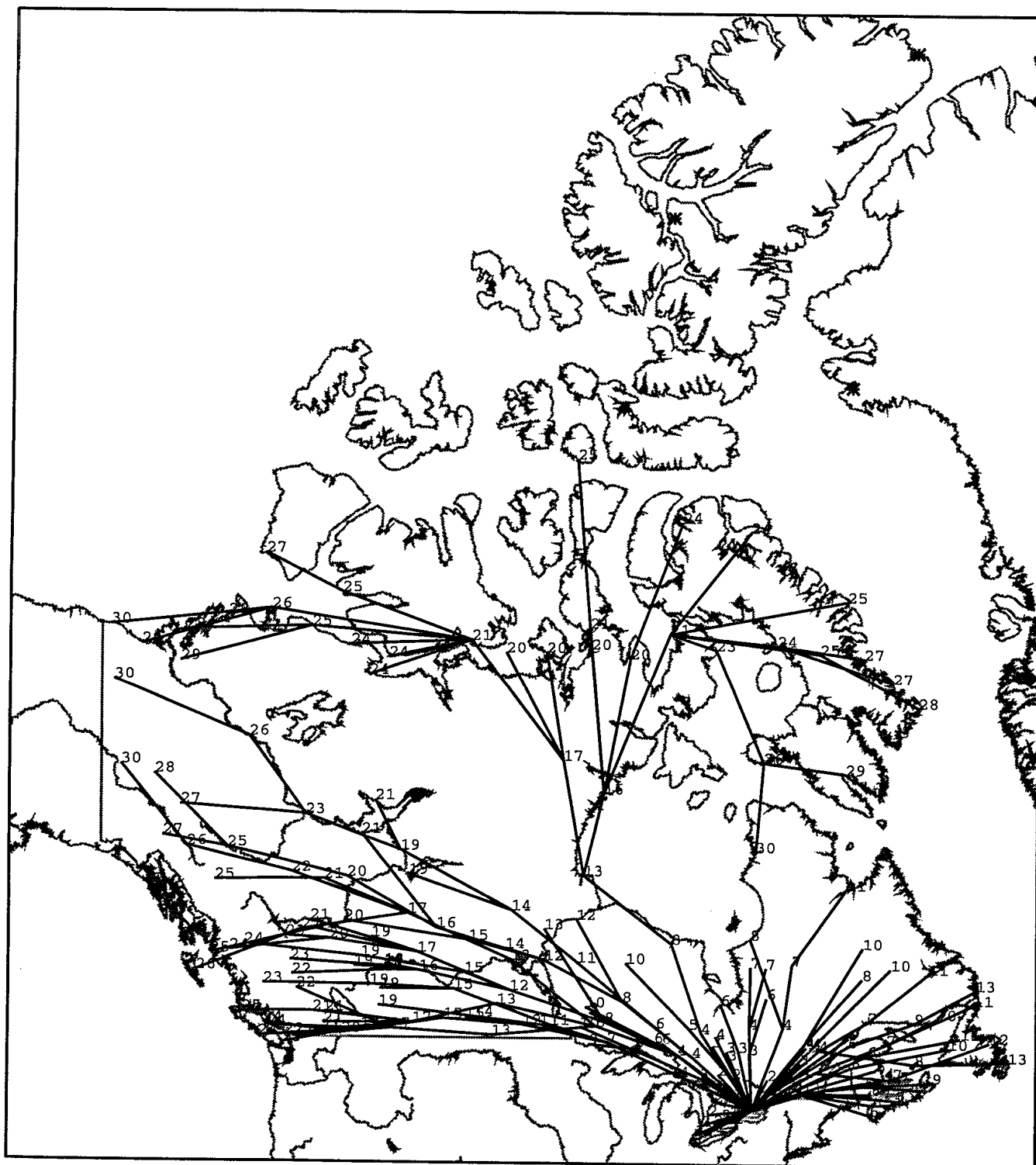
71. The results of the Model indicated that there is a minor difference in the coverage provided by the helicopters under VFR flight conditions. In VFR conditions, the CH-113 can reach every refuelling site, while the Bell 412SP helicopter can reach all but one refuelling site. Under IFR conditions, there is a significant difference between the helicopters. Operating with IFR rules, the CH-113 can reach all but the three most northern refuelling sites, in contrast to the Bell 412SP which cannot reach any of the refuelling sites in the Northwest Territories or the Yukon Territory. Also, to arrive at the extreme refuelling sites that the Bell 412SP can reach takes eight to nine hours longer than for the CH-113. The IFR

- 18 -

results for the Ch-113 and Bell 412SP helicopters are shown in Figures 1 and 2, respectively. Asterisks in the Figures indicate refuelling sites that could not be reached.

72. The application of the SAR Helicopter Transit Model to the comparison of the CH-113 and Bell 412SP helicopters has demonstrated that the Model is very effective for relating helicopter flight characteristics to coverage and transit performance.

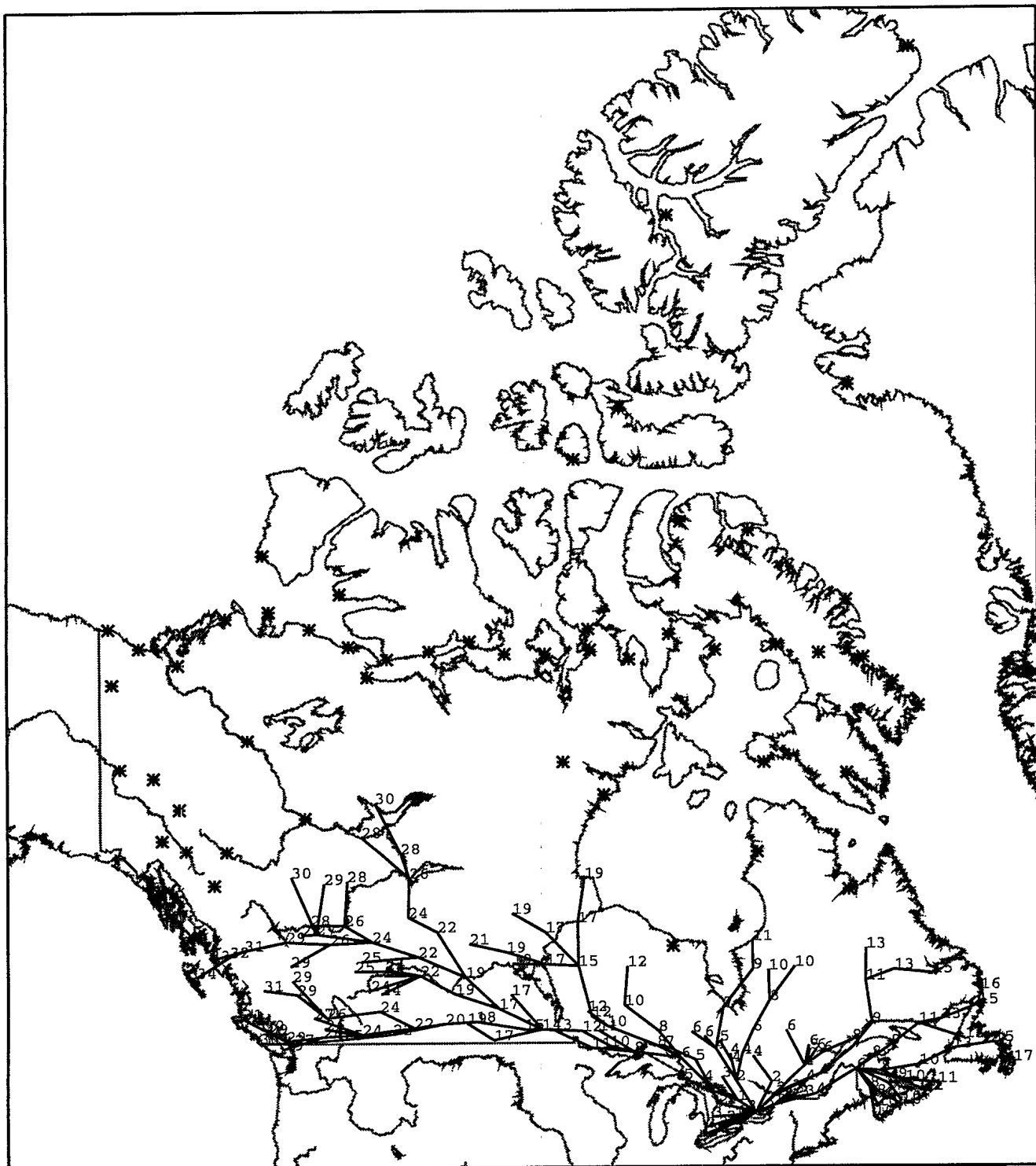
- 19 -



CH-113 IFR Coverage from Trenton

FIGURE 1 : CH-113 IFR TRANSIT TIME AND COVERAGE

- 20 -



Bell 412SP IFR Coverage from Trenton

FIGURE 2 : BELL 412SP IFR TRANSIT TIME AND COVERAGE

ANNEX A
DAOR RESEARCH NOTE 94/8
NOVEMBER 1994

USER GUIDE FOR SAR HELICOPTER TRANSIT MODEL

1. The SAR Helicopter Transit Model consists of two computer programs: the Helicopter Transit Program and the Mapping Program. The first program determines the shortest transit path and time for the SAR helicopters to reach each refuelling site operating under one of two flying conditions, Instrument Flying Regulations (IFR) or Visual Flying Regulations (VFR). The second program creates a map from files containing map data for Canada, then overlays the results generated from the first program on top of the map.

OPERATING THE TRANSIT PROGRAM

2. The Helicopter Transit Program is written in SUN FORTRAN labelled as "base.for". The compiled version of the program is created by running the SUN FORTRAN compiler, typing:

```
f77 -o base base.for
```

This generates a file named "base" which is the file that is executed.

3. Before starting the program, one must make sure the required data files and the parameters to be passed to the program are prepared. The program requires a data file containing the names and positions of the available refuelling sites, and a data file identifying the sites which will be used as helicopter bases. The names of the files are identified as parameters during execution of the Helicopter Transit Program, "base".

4. The refuelling sites data file is structured such that each line contains the name, latitude position (degrees and minutes) and longitude position (degrees and minutes) for each refuelling site available to helicopters. The file is formatted, so, data must be in the proper positions to be entered correctly. The FORTRAN format followed is "A15,18X, I2,I2,3X,I3,I2". An extract of a refuelling sites data file is shown in Figure A-1.

5. The structure of the data file used to identify the helicopter bases will be explained later.

WINDSOR	0421629008257300
CHATAM ONT	0421824008204500
ST THOMAS	0424611008106200
SARNIA	0425958008218320
LONDON	0430208008109140
BRANTFORD	0430753008020340
HAMILTON	0431019007955540
ST CATHARINES	0431130007910190
WATERLOO-GUELPH REGIONAL	0432732008023050
TORONTO/LESTER B. PEARSON	0434038007937510
GODERICH	0434601008142380
YARMOUTH	0434937006605190
TORONTO/BUTTONVILLE	0435144007922130
OSHAWA	0435522007853430
TRENTON	0440708007731420
KINGSTON	0441331007635570
PETERBOROUGH	0441348007821490
DIGBY NS	0443255006547000
BROCKVILLE	0443820007545030
HALIFAX/SHEARWATER	0443823006330000
WIARTON	0444445008106260
HALIFAX INTL	0445251006330330
MUSKOKA	0445829007918130
GREENWOOD	0445904006455030
BROMONT	0451727007244310
ST-JEAN	0451740007316540
SAINT JOHN	0451858006553270
OTTAWA INTL	0451921007540100
SHERBROOKE	0452617007141280

FIGURE A-1: REFUELLING SITES DATA FILE SAMPLE

6. The program, "base", requires the following parameters:
 - a. the file name of the refuelling sites data file,
 - b. whether the flying condition is IFR or not,
 - c. the file name for the output file,
 - d. the file name of the helicopter bases data file,
and
 - e. for each helicopter type to be modelled: the
range, speed, number of bases, and the names of bases.

File names must be the complete name, including extensions, in proper upper and lower case. The IFR flying condition is a logical response, i.e. *.TRUE.* or *.FALSE.* Speed and range are in units of knots and nautical miles, respectively. The program pursues a cycle of requesting the range, speed, number of bases, and the names of bases for each helicopter type. The base names must be names of refuelling sites contained in the refuelling sites data file. (A sample of a refuelling sites data file is shown in Figure A-1.) The input cycle is stopped when a non-numeric input, such as "stop", is entered at the prompt for helicopter range.

7. The responses to the requested parameters can be entered from the keyboard or through a parameter file. When the program is initiated, the user is immediately requested to identify the name of the parameter file to be used. If no name is entered, i.e. just "enter" or carriage return is hit, the user will be prompted to enter each of the parameters identified above. If a parameter file is identified, the program will execute without further input from the user.

8. The following is an example of an interactive initiation sequence of the helicopter transit program. Program prompts are in italics and user responses are enclosed in pointed brackets, *< >*. A carriage return is entered after each user response. Note that *<CR>* represents a sole entry of a carriage return.

```

<base>
Input parameter file name :
<CR>
Input refuelling sites data filename :
<IFRSites.dat>
IFR (.TRUE. or .FALSE.) :
<.TRUE.>
Output filename :
<test1.out>
Helicopter bases data filename :
<bases2.dat>
Range of Helicopter :
<300>
Speed of Helicopter :
<120>
No. of bases for this helicopter type :
<1>
base 1 for this helo type :
<TRENTON>
Range of Helicopter :
<stop>

```

The contents of a parameter file to accomplish the same input specifications is shown in Figure A-2.

```

IFRSites.dat
.TRUE.
test1.out
bases2.dat
300.
120.
1
TRENTON
stop

```

FIGURE A-2: SAMPLE TRANSIT PROGRAM PARAMETER FILE

9. An example of running the transit program using a parameter file, "test1.parm", would be:

```

<base>
Input parameter file name :
<test1.parm>

```

Note, the parameter file must be in the same directory as the transit program in order for the program to locate the file.

10. The file specified as containing the list of helicopter bases to be used in the iteration (parameter "d" above) must identify all the refuelling sites that will be used as bases. The input sequence to initiate the transit program identifies which helicopter types are to be located at which bases. Helicopters must be allocated to each base specified in the base file and the name of every base allocated helicopters must be contained in the base file. The base file is structured with a comment line at the beginning, the number of bases to be used (format I2), and the names (format A15) of the refuelling sites to be used as base. Note that upper and lower case usage when specifying base and refuelling site names is important. Site names must match exactly. An example of the contents of the base file used to support the transit program initiation sequence shown above is contained in Figure A-3.

11. After starting the program and responding to the input requests, the program verifies that there is no inconsistency between the bases identified in the data file, those specified for the helicopter types and there exists a corresponding refuelling site. If any problem is detected, a message is output to the user indicating the

problem, and a suggestion to correct the problem and restart to program. After the user is given the message, the program is stopped.

```
** Comment : Test run using Trenton as base.  
1  
TRENTON
```

FIGURE A-3: EXAMPLE OF BASE DATA FILE (bases.dat)

12. If no problem is detected, the transit program proceeds to determine the optimal path and transit times to arrive at each refuelling site that can be reached by the collection of helicopter types and bases. The results are stored in the designated output file. An example of a transit program output file is shown in Figure A-4.

OPERATING THE MAPPING PROGRAM

13. The source code for the mapping program is contained in a file labelled "sarmap.for", and it is also written in SUN FORTRAN. The compiled version of the program is generated by running the compiler:

```
f77 -o sarmap sarmap.for
```

This produces the executable file "sarmap".

14. The sarmap program is derived from a general map producing program. As such, there are a number of features and input requests that are not usually used to produce a map output for the SAR Helicopter Transit Model. The features have been left as part of the map program to maintain the extra flexibility of the program should it ever be required. The operating procedures described in this document will deal with running the map program in the usual manner to produce a helicopter transit map. It will be left to the user to examine the "sarmap.for" source code to identify the functions of the extra features not described in this documentation.

```

1
120
1.0
300
171
42.266666 -82.950005 4
42.299999 -82.066666 3
42.766666 -81.099998 2
42.983334 -82.300003 3
43.033333 -81.150002 2
43.116665 -80.333336 2
43.166668 -79.916672 1
.
.
.
60.833332 -115.766670 28
62.450001 -114.433334 30
50
54.9833 -85.4333
58.0833 -68.4167
58.4167 -130.017
60.0333 -77.2500
60.1167 -128.817
60.1667 -132.733
60.7000 -135.050
.
.
.
79.9833 -85.8000
82.5167 -62.3167
2
43.1167 -80.3333
42.2667 -82.9500
2
43.1667 -79.9167
42.3000 -82.0667
.
.
.
2

```

FIGURE A-4: EXAMPLE OF TRANSIT PROGRAM OUTPUT FILE

15. In its routine usage, the map program takes the output from the transit program and overlays it onto a user-defined map region. The user identifies the data file from the transit program. The user can specify up to two lines of text, 41 characters each, to be used as a title for the map figure. The program utilizes a default data file, "g1.src", to create the map section. The final result of the map program is a postscript file named, "sarmap.out".

16. The "g1.src" data file contains the data required to display the map in low resolution mode. The uncompressed file occupies approximately three and one half megabytes of disk space. The file is organized as a collection of groups of latitude and longitude positions which represent points on the coastline of the map to be constructed. The points are translated into positions on the map grid and lines are drawn between the points to form the coastlines for the map. A sample of the contents of the map data file is shown in Figure A-5.

17. As with the transit program, the map program can be operated with a parameter file to specify the required inputs, or interactively with the keyboard. The required inputs for the map program are:

- a. the name of the parameter file (if used),
- b. an overlay file name,
- c. are there input points and lines (y or n),
- d. refuelling points file name,
- e. a file name of lines,
- f. a file name of points,
- g. a file name of titles,
- h. a file name of panels,
- i. a file name of circles,
- j. the map window (latitude and longitude range),
- k. the projection type,
- l. the map resolution,
- m. the plot size (percentage of output page),

- n. the first line of the title, and
- o. the second line of the title.

18. In the default usage mode of the sarmap program, only parameters a, c, d, j, k, l, m, n, and o are provided with responses. The other parameters are left blank, carriage returns are supplied as responses. If a file name is provided for input "a", all the other input parameters will be taken from the file without requiring additional user action.

19. The output file from the transit program is identified to the sarmap program by answering "y" to input request "c" and specifying the name of the file in response to input request "d". The map window (parameter "j") is defined by the northern latitude boundary, the southern latitude boundary, western longitude boundary and the eastern longitude boundary. South latitudes and west longitudes are entered as negative values. A suitable map window for displaying the Canadian SAR area is defined by 83 and 40 degrees for the latitude boundaries, and -150 and -50 degrees as longitude boundaries. The projection type can be lambert or mercator. All SAR maps produced for DAOR projects in the recent past have used the mercator project. The mercator projection is recommended as the default response when running the sarmap program. The map resolution parameter defines the density of points used to draw the coastlines of the map. The high resolution requires a very large data file that currently is stored only on tape. The low resolution has been found to produce an acceptable map and does not require a tape mount. Low resolution is the recommended default response. The output size denotes the proportion of a standard page that will be used to display the map. A 50 percent response results in the map being scaled to use half the page. It is suggested that all maps use the full page (100 percent response). The user can specify up to two lines of text to be used as the title of the map figure and will be centred at the bottom of the map. If the user chooses not to enter a title, a title will be created from the name of the refuelling sites input file identified in parameter "d".

20. When a title is to be created from the input file name, character strings separated by periods are transformed into separate words by replacing the periods with spaces. This word string is then added to the string, "TRANSIT TIME and COVERAGE", to form the title. This title is then displayed centred at the bottom of the map figure. For example, if an input file named "LABRADOR.VFR" were used and no title was specified, a default title of "LABRADOR VFR TRANSIT TIME and COVERAGE" would be created and displayed on the map figure. As there is no limitation to the number of characters used to name a file, input file names can easily be used to identify the file and form the map title.

```

g1.src
VM      WW      0000000001000011  0000000000000000
286500 0376020 5
286680 0376020 2
286620 0376200 1
286500 0376260 2
286620 0376320 2
286680 0376200 2
286800 0376320 1
286980 0376320 2
286980 0376440 1
287040 0376500 2
287460 0376620 3
287460 0376740 2
260760 0383700 5
9999999999999999
VM      WW      0000000001000011  0000000000000000
260760 0383700 5
260340 0383400 2
260220 0383100 2
260280 0383040 1
260340 0383040 1
260340 0382980 2
260280 0382920 2
260220 0382980 2
260160 0383160 2
260100 0382680 1
246540 0388920 2
246420 0388740 5
9999999999999999
CB      WW      0000000001000011  0000000000000000
282120 0370500 5
282360 0370620 2
282240 0370680 1
282120 0370680 2
282060 0370620 1
281820 0370680 2
281700 0370620 1
281580 0370440 1
281400 0370440 3

```

FIGURE A-5: SAMPLE OF THE "g1.src" MAP DATA FILE

21. An example of an interactive initiation of the sarmap program is shown below. Program output (parameter requests) are shown in italics and user responses are contained in pointed brackets, < >. A carriage return is entered after each user response. <CR> represents a sole entry of a carriage return.

```

<sarmap>
Input parameter file name
<CR>
Input overlay file name
<CR>
Do you have files of input points and lines?
<y>
Input file name of refuelling points
<test1.out>
Input file name of lines
<CR>
Input file name of points
<CR>
Input file name of titles
<CR>
Input file name of panels
<CR>
Input file name of circles
<CR>
Input window (north latitude, south latitude
               west longitude, east longitude)
<83>
<40>
<-150>
<-50>
Input Proj (lambert = 1, mercator=2)
<2>
Input resolution (low=1, high = 2 (requires tape mount))
<1>
Input output plot size (as a % of full size)
<100>
Input title first line (default => file name)
<HELO COVERAGE FROM TRENTON>
Input title second line (default => file name)
<SPEED: 120 KT, RANGE 300 NM, IFR>

```

The sarmap program would now proceed to produce the postscript output file and store it as "sarmap.out".

22. An example of using a parameter file to initiate the sarmap program would be:

```
<sarmap>  
Input parameter file name  
<sarmap.parm>  
Input overlay file name  
<CR>
```

The sarmap program would now use the parameter file "sarmap.parm" to obtain responses to the other input requests and would proceed to develop the output file as in the previous example. A parameter file corresponding to the interactive input defined above is shown in Figure A-6.

```
y  
test1.out  
  
83  
40  
-150  
-50  
2  
1  
100  
HELO COVERAGE FROM TRENTON  
SPEED: 120 KT, RANGE: 300 NM, IFR
```

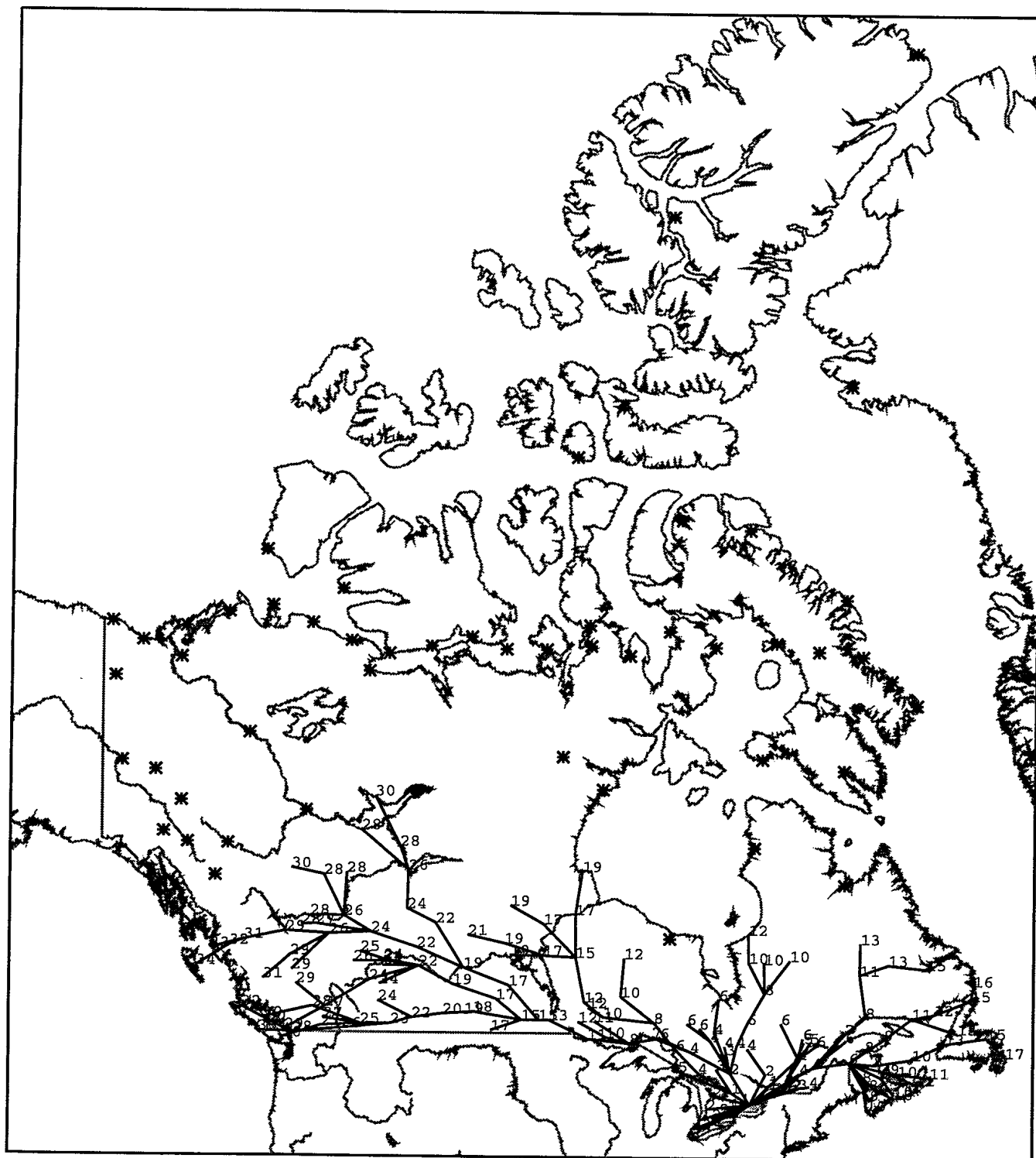
FIGURE A-6: AN EXAMPLE OF A SARMAP PARAMETER FILE

VIEWING AND PRINTING THE SARMAP OUTPUT FILE

23. Within the Sun workstation operating system, there is a utility called "pageview" which permits a postscript file to be displayed as it would appear on a printed page. A new window is created and a representation of printed page is displayed. From this window, there is an option under the "file" menu which permits the output displayed to be sent to a printer for a paper copy to be produced. An example of the final product of the sarmap program is shown in Figure A-7.

24. To view the output contained in the file "sarmap.out", enter the following command:

```
pageview sarmap.out
```



120KT, 300NM, IFR Coverage from Trenton

FIGURE A-7: EXAMPLE OF SARMAP OUTPUT

ANNEX B
DAOR RESEARCH NOTE 94/8
NOVEMBER 1994

SAR HELICOPTER MODEL PROGRAM LISTINGS

1. The following pages contain a listing of the FORTRAN source code for the helicopter transit program "base.for" and the map generation program "sarmap.for".

```

!      backup in : PGM.tar
c*****
c
c      Program Name : base.for
c
c      Function :   This program find the shortest path of refuelling sites in
c                  terms of time to get to that sites.
c                  Several factors determine the shortest time getting to a
c                  site. They are the given reference sites, the available
c                  helicopter type at these reference sites and the type of
c                  simulation we use (IFR or VFR). The resulting reachable
c                  sites at shortest possible time and the location of the
c                  unreachable sites will be stored in a output file for
c                  further process.
c
c      Input :  pname      parameter file, optional. Defaulted to be standard
c                  input.
c                  in_fname, number of points in in_fname, whether it
c                  is a IFR simulation, out_fname, ref_fname as well
c                  as the helicopters' informations are stored.
c                  in_fname      input data file. It contains the name and the
c                  location of refuelling sites. When not given,
c                  flsr3.dat(for IFR) or jetf.dat(for VFR) will be
c                  used depending on the type of simulation being used.
c                  ref_fname     reference point file. It contains the number of
c                  reference points to be used in the run in the first
c                  line followed by the name and location of the
c                  reference points. Default is RefPt.dat
c
c      Output : out_fname  output file, it stores the shortest time to reach a
c                  sites (defined by its location), the location of
c                  unreachable sites and the paths forming the shortest
c                  time path is stored in out_fname.
c
c      Note : The helicopter information given in pname includes the reference
c              number of the reference point that have that type of helicopter
c              available. For a given reference points, we know the range and
c              speed of the helicopter, and thus the time to to reach difference
c              sites from that reference point.
c
c              We are assuming a one to many relationship between the reference
c              points and the helicopter types.
c              i.e. there can only be one helicopter type serving a reference
c              point, but different reference points can be using the same
c              helicopter type
c
c              For each sites, the path starting from a reference point giving
c              shortest time will be shown. It is possible to have two parents
c              for a given site in the output being generated because the this
c              site might be in the shortest path of another site while it is
c              having another shortest path.
c
c              The maximum number of reference points is 10, if a larger number
c              of reference points is tested, should modify the data structure
c              to handle extra reference points.
c
c      Date : 9 June 1994
c*****
c*****
c
c              Other Variables Used
c
c      nam      - contains the name of all the refuelling sites in string
c      b_name    - contains the name of reference sites (the base)
c      ltd, ltm, lgd, lgm, lat and lon are those for other sites
c      d         - matrix of distances from one site to the other
c      tim       - matrix of shortest time needed from reference points of
c                  same helicopter type to other sites if a path exists
c      ref       - the index of reference sites with respect to the order
c                  of sites being read from file, a base is supposed to be
c                  found in the file
c      nref      - the total number of reference sites
c      h_tim     - the time matrix of the final analysis with different
c                  helicopter type and thus different reference points
c      h_ref     - the corresponding ref and nref for a specific helicopter

```

```

c      h_nref      type
c      lnk         - index of parent originated from each reference points
c      range       - range of different helicopter types
c      spd         - speed of different helicopter types
c      nheli       - number of helicopter type that is available
c      heli_ref-    matrix contains the availability of each helicopter type
c                  by storing the index of reference point they are available
c      isIFR       - true if the run is in IFR condition.
c      max_ref     - maximum number of reference sites allowed and therefore
c                  the maximum number of helicopter types being read
c
c      modified by GLC 200ct94
c *****
character*15 nam(500), b_name(10), h_b_name(10,10)
character*1 bk
character*72 in_fname, out_fname, pfname, ref_fname
real lat(500),lon(500),d(500,500),mds,tim(500,10),h_tim(500,10)
integer loc(500),ref(10),plac(500),lnk(500,10),hit(500,500)
integer nr(500)
real range(10), spd(10) !the two together identify a helicopter type
integer nheli, heli_ref(10,10) !storing information relating to ref.pts
integer h_nref, h_ref(10)
logical isIFR
data pi/3.1415926536/
data bk/' ', isIFR/.false./, max_ref/10/
data h_b_name/100*' '/
common /heli_info/range,spd,nheli,heli_ref,max_ref
logical find_rg_sp

c *****
c                               Main program
c *****
write(*,*) "Input parameter file name "
read(*,900) pfname
900 format(a)
if ( pfname .eq. " " ) then
  i_file = 5
else
  i_file = 4
  open (i_file,file=pfname,form="formatted")
endif

c=====
c Read from parameter file, if exists. Otherwise from stdin
c=====
920 write(*,*) "Input refuelling sites data filename :"
read (i_file,*,end=930,err=930) in_fname

c following lines modified by GLC 200ct94
c write(*,*) "Number of points in input file :"
c read (i_file,*,end=930,err=930) nn

write(*,*) "IFR (.TRUE. or .FALSE.) :"
read (i_file,*,end=930,err=930) isIFR
write(*,*) "Output filename :"
read (i_file,*,end=930,err=930) out_fname
write(*,*) "Helicopter bases data filename :"
read (i_file,*,end=930,err=930) ref_fname
goto 940
930 write(*,*)
write(*,*) " *****"
write(*,*) " Error reading parameter file : ",pfname
goto 1000

940 continue
if ( in_fname .eq. " " ) then
  if ( isIFR ) then
    in_fname = "IFRSites.dat"
c      nn=222 ! no. of points in flsr3.dat
  else
    in_fname = "VFRSites.dat"
c      nn=326 ! no. of points in jetf.dat
  endif
endif
if ( out_fname .eq. " " ) then
  if ( isIFR ) then
    out_fname = "FLIGHT.IFR"

```

```

        else
            out_fname = "FLIGHT.VFR"
        endif
    endif
    if ( ref_fname .eq. " " ) then
        ref_fname = "RefPt.dat"
    endif
    open (1,file=in_fname, form="formatted")
    open (2,file=ref_fname,form="formatted")
    open (3,file=out_fname,form="formatted")

    nheli = 0
760 continue
    write(*,*) "Range of Helicopter :"
    read (i_file,*,end=700,err=700) range(nheli+1)
    write(*,*) "Speed of Helicopter :"
    read (i_file,*,end=720,err=720) spd(nheli+1)
    write(*,*) "No. of bases for this helo type :"
    read (i_file,*,end=720,err=720) nref
    do 750 i=1,nref

c        write(*,*) i,"th index of reachable reference points :"
c        read (i_file,*,end=720,err=720) heli_ref(nheli+1,i)
c        following lines added by GLC 200ct94
        write(*,*) "base ",i," for this helo type :"
        read (i_file,745,end=720,err=720) h_b_name(nheli+1,i)
745    format(a15)

750 continue
    nheli = nheli + 1
    goto 760
720 write(*,*)
    write(*,*) " *****"
    write(*,*) " Incomplete or invalid information from Helicopter ",
    &         nheli+1
    goto 1000
700 continue
    write(*,*) "Helicopter types available :",nheli

c=====
c Read from file containing informations of the Reference
c points. The first line of file is comment which is skipped
c Base reference index from the file is initialized be to 0
c=====
    read (2,*,err=310) !skip the first line (comment line)
    read (2,*,err=310) nref
    do 300 i=1,nref
        read (2,600,err=310) b_name(i)
600    format(a15)
        ref(i) = 0
300 continue
    goto 320
310 write(*,*)
    write(*,*) " *****"
    write(*,*) " Error from Reference Point File : ",ref_fname
    goto 1000

c =====
c Initializing variable for either IFR or VFR simulation
c
c mds - mds*range defines the minimum distance between two
c       sites
c nds - maximum number of sites in a path
c tps - time per stop
c =====
320 continue
    erad=10800./pi
    mds=0.15
    nds = 500
    if ( isIFR ) then
        tps = 1.0
    else
        tps = 0.5
    endif

```



```

c =====
c Read from input file, the data file containing the refuelling
c sites. Index them according to the sequence read in. At the
c same time, indexes of sites having the same name are taken to
c be the references themselves. The position of references are
c indicated by array ref, which is the index of them in the
c data list
c
c Note that site with exactly the same name as a reference
c will be identified as the reference immediately and there
c will be no more searching on that base
c
c The following section has been modified by GLC 200ct94
c =====
      i=0
617 i=i+1
      read (1,607,end=625,err=627) nam(i),ltd,ltm,lgd,lgm
      lat(i)=(ltd+ltm/60.)*pi/180.
      lon(i)=(lgd+lgm/60.)*pi/180.
607 format (a15,18x,i2,i2,3x,i3,i2) !jetf.dat & flsr3.dat
      write(*,*) "Read : ",nam(i),ltd,ltm,lgd,lgm
      do 400 j=1, nref
        if (ref(j) .EQ. 0) then
          if (name_req(b_name(j),nam(i))) then
            write(*,*) i," th index ",nam(i)
            ref(j) = i
            goto 618
          endif
        endif
400 continue
618 do 619 j=1,nheli
      k=1
621 if (h_b_name(j,k) .eq. ' ') goto 619
      if (name_req(h_b_name(j,k),nam(i))) then
        heli_ref(j,k)=i
      else
        k=k+1
        goto 621
      endif
619 continue
      goto 617
625 nn=i-1
      write (*,*)
      close (1)
      goto 630
627 write(*,*)
      write(*,*) " *****"
      write(*,*) " Error reading Data File : ",in_fname
      goto 1000

c =====
c Check if there is any reference point that cannot be found in
c the data set and if there is at least one helicopter type
c available in the bases.
c
c If any of the bases could not be found from the reference pts.
c data file, or if there is a helicopter type matching the base
c then no further processing is done.
c =====
630 continue
      do 500 i=1,nref
        if (ref(i) .EQ. 0) then
          write(*,*)
          write(*,*) " *****"
          write(*,*) " Base not found :",b_name(i)
          goto 1000
        endif
500 continue
      do 520 i=1,nref
        write(*,*)
        write(*,*) " ** Reference ",i
        write(*,*) " For reference ",b_name(i),ref(i)
        if (find_rg_sp(ref(i),rang,sp)) then
          write(*,*) " Helicopter range : ",rang
          write(*,*) " speed : ",sp
        else
          write(*,*)

```

```

        write(*,*) " *****"
        write(*,*) " No helicopter available at base ",b_name(i)
        goto 1000
    endif
520 continue

c =====
c Start the path and shortest time finding
c
c It is done once run for each helicopter types becasue their
c range and speed are different.
c In each iteration, reference points using that helicopter
c type will be considered as bases, the remaining references
c will not used as base but their path will not be found.
c
c The algorithm is seperated into 3 parts, the first part is
c to find the time between each pair of sites. The second part
c is to find the shortest time path
c The third part is done after we got part 1 and 2 for each
c helicopter type, and the shortest time is found among them
c for each site
c =====
        write(3,*) nheli
        do 510 i_heli=1,nheli
            rang = range(i_heli)
            sp = spd(i_heli)
            write(*,*)
            write(*,*) "*** Helicopter",i_heli
            write(*,*) " Range : ",rang," and speed : ",sp

c =====
c h_nref - the number of reference sites using that helicopter
c type
c h_ref - array of index of all reference sites
c
c If no reference site is using that helicopter type, it will
c be ignored
c =====
            h_nref = 0
            do 130 i=1,nref
                do 129 j=1,max_ref
                    if (heli_ref(i_heli,j) .EQ. 0) goto 130
                    if (heli_ref(i_heli,j) .EQ. ref(i)) then
                        h_nref = h_nref + 1
                        h_ref(h_nref) = ref(i)
                        write(*,*) " Reference : ",b_name(i),ref(i)
                    enddo
                enddo
            continue
129 continue
130 write(*,*) " There are ",h_nref," ref. pts"
            isp=sp
            write(3,131) isp
131 format(15)
            write(3,132) tps
132 format(f3.1)
            irg=rang
            write(3,131) irg

            if (h_nref .EQ. 0) goto 510

c =====
c To clear the variable and re-initialize them
c =====
            nq=0
            do 504 i=1,nn
                do 503 j=1,500
                    d(i,j) = 0
                    hit(i,j)=0
2503 continue
                nr(i)=0
                loc(i)=0
                plac(i)=0
                lnk(i,i_heli)=0
504 continue

```

```

c =====
c Part 1      Finding shortest time between any 2 sites for
c              a given helicopter type
c =====
      do 50 i=1,nn
      nj=1
      nr(i)=0
      do 45 j=1,nn
      if (i.eq.j) go to 45
      dj=dst(pi/2.-lat(i),pi/2.-lat(j),lon(i)-lon(j),erad)
      if (dj.gt.rang) go to 45

c =====
c Added for IFR

      if ( .NOT. isIFR ) goto 60
c      modification below by GLC to do fued wing runs 45 Aug 1994
c      if (dj.lt.mds*rang) go to 45
      do 56 k=1,nn
      if (k.eq.j) go to 56
      dk=dst(pi/2.-lat(j),pi/2.-lat(k),lon(j)-lon(k),erad)
      if (dk.lt.100.) go to 56
      xj=rang-dj
      if (dk.gt.xj) go to 56
      go to 60
56      continue
      go to 45
60      continue
c =====

      d(i,nj)=dj/sp
      hit(i,nj)=j
      nr(i)=nj
      nj=nj+1
      if (nj.gt.nq) nq=nj
      if (nj.gt.nds) then
      write (*,*) 'entered gt nds ',i,nj,nq
      nj=nj-1
      xj=10000.
      kj=0
      do 43 k=1,nj
      if (d(i,k).lt.xj) then
      xj=d(i,k)
      kj=k
      endif

c =====
c Added for IFR

      if ( .NOT. isIFR ) goto 43
      if (d(i,k)*sp.lt.mds*rang) then
      write(*,*) ' found ',nam(k),i,k
      do 42 m=k,nds-1
      d(i,k)=d(i,k+1)
42      continue
      go to 45
      endif

c =====
43      continue
      if (kj.eq.0) write (*,*) ' i,kj ',i,kj
      write (*,*) i,' Exceeds ',nds,' at 45-1 to ',j,kj
      do 44 k=kj,nds-1
      d(i,k)=d(i,k+1)
      hit(i,k)=hit(i,k+1)
44      continue
      endif
45      continue
50      continue
      write (*,*) ' nq ',nq
      do 160 i=1,nn
      nj=nr(i)
      if (nj.le.1) go to 160
      do 150 j=1,nj-1
      xmn=d(i,j)
      n=j
      do 140 k=j+1,nj

```

```

        if (d(i,k).lt.xmn) then
            xmn=d(i,k)
            n=k
        endif
140    continue
        xj=d(i,j)
        d(i,j)=xmn
        d(i,n)=xj
        nk=hit(i,j)
        hit(i,j)=hit(i,n)
        hit(i,n)=nk
150    continue
160    continue

c      =====
c      Added for VFR
c      This is part of the original program and makes Vancouver the
c      only reachable site for Comox and Williams Lake the only reachable
c      site from Vancouver, when helicopters are operating VFR
c      It is unknown why this was done.
c      Section commented out by GL Christopher 24 Aug 1994
c
c      if ( .NOT. isIFR ) then
c          do 167 i=1,2
c              if (i.eq.1) then
c                  nj=138
c                  nk=nr(nj)
c              endif
c              if (i.eq.2) then
c                  nj=120
c                  nk=nr(nj)
c              endif
c              do 165 j=1,nk
c                  if (i.eq.1.and.hit(nj,j).eq.120) then !Vancouver
c                      hit(nj,1)=120
c                      d(nj,1)=d(nj,j)
c                      nr(nj)=1
c                      go to 167
c                  endif
c                  if (i.eq.2.and.hit(nj,j).eq.190) then !Williams Lake
c                      hit(nj,1)=190
c                      d(nj,1)=d(nj,j)
c                      nr(nj)=1
c                      go to 167
c                  endif
c165          continue
c167          continue
c          write (*,*) ' hit from ',nam(138),' to ',nam(hit(138,1))
c          write (*,*) ' and from ',nam(120),' to ',nam(hit(120,1))
c      endif
c      =====

c      =====
c      Part 2      Finding shortest time and path for each site
c                  for a given helicopter type
c      =====
        njq=0
        do 100 k=1,h_nref
            loc(1)=h_ref(k)
            nj=1
            ij=loc(1)
            xt=0.
            do 75 i=1,nn
                tim(i,k)=0.
75          continue
80          continue
            na=1
            =====
c          The path searching starts from here and keep on until
c          it reach a reference point or a point which is already
c          in the path
c          For each site, the parent of it will be stored in lnk,
c          one of each helicopter type being encountered.
c          The parent found should be the previous site in the path
c          giving the minimal time

```

```

c =====
85  continue
    nk=nr(ij)
    if (nk.eq.0) go to 98
    do 90 m=na,nk
        i=hit(ij,m)

c  Following line added for debugging purposes
c  GLC Oct 94
c
c  write (*,*) "ij m i nk nj",ij,m,i,nk,nj

    do 86 j=1,nj
        if (i.eq.loc(j)) go to 90
86  continue
    do 87 j=1,h_nref
        if (i.eq.ref(j)) go to 90
87  continue
    nj=nj+1
    loc(nj)=i
    plac(nj)=m
    xt=tim(ij,k)+tps+d(ij,m)
    xm=tim(i,k)
    xt=tim(ij,k)+tps+d(ij,m)
    if (lnk(i,i_heli).eq.0) lnk(i,i_heli)=ij
    if (tim(i,k).eq.0.) tim(i,k)=xt
    if (xt.lt.tim(i,k)) tim(i,k)=xt
    do 89 j=1,k
        if (tim(i,j).eq.0.) go to 89
        if (tim(i,j).lt.xt) go to 93
89  continue
    lnk(i,i_heli)=ij
93  continue
    if (xm.eq.0..or.xm.gt.xt) then
        ij=i
        go to 80
    else
        na=m+1
        if (na.gt.nr(ij)) go to 90
        nj=nj-1
        go to 85
    endif
90  continue
98  continue
    nj=nj-1
    if (nj.lt.1) go to 100
    na=plac(nj+1)+1

c  Following lines added for debugging purposes
c  GLC Oct 94
c
c  write (*,*)
c  write (*,*) "*** at statement 98 ***"
c  write (*,*) "nj na loc(nj+1)",nj,na,loc(nj+1)
c  write (*,*) "nr(loc(nj+1)) loc(nj)",nr(loc(nj+1)),loc(nj)
c  write (*,*) "nr(loc(nj))",nr(loc(nj))
c  write (*,*) "ij m nk",ij,m,nk
c  write (*,*)

c  Following line modified to correct error
c  GLC Oct 94
c
c  if (na.gt.nr(loc(nj+1))) go to 98
c  if (na.gt.nr(loc(nj))) go to 98
c  ij=loc(nj)
c  go to 85
100 continue

c =====
c  Now we have all the shortest time and path between two
c  sites and they are stored in tim and lnk respectively.
c  tim is a matrix storing time for each site starting from
c  different reference point, the smallest time will be
c  the time needed defined by the path found above.
c
c  h_tim - the time to reach each site for each helicopter

```

```

c          type
c  =====
do 110 i=1,nr
do 226 j=1,nref
  if (i.eq.ref(j)) go to 110
226 continue
do 171 m=1,h_nref
  if (tim(i,m).gt.0.) then
    xj=tim(i,m)
    do 181 j=m,h_nref
      if (tim(i,j).eq.0.) go to 181
      if (tim(i,j).lt.xj) xj=tim(i,j)
181    continue
    h_tim(i,i_heli)=xj
    goto 110
  endif
171 continue
110 continue
510 continue !for different helicopter with varying range and speed

```

```

c  =====
c  Part 3      Finding shortest time and path for each site
c              among all the available helicopter types
c  =====
c  =====
c  nj          number of reachable sites
c  ncg         number of unreachable sites
c
c  nj and ncg are found and the locations of corresponding
c  sites are stored in output file
c  smallest time in h_tim is found, and so we know which
c  helicopter type we will be using for a specific site by
c  getting the the parent from lnk directly
c  A negative sign is used as an indicator for sites used
c  in the shortest path.
c  =====
nj=0
ncg=0
do 200 i=1,nr
do 225 j=1,nref
  if (i.eq.ref(j)) go to 200
225 continue
do 170 m=1,nheli
  if (h_tim(i,m).gt.0.) go to 175
170 continue
  ncg=ncg+1
  loc(ncg)=i
  write(*,*) ' Can not get to ',nam(i)
  go to 200
175 continue
  nj=nj+1
  xj=10000.
  do 180 j=1,nheli
    if (h_tim(i,j).eq.0.) go to 180
    if (h_tim(i,j).lt.xj) then
      xj=h_tim(i,j)
      j_next = j
    endif
180 continue
    i_next = i
177 continue
    if (lnk(i_next,j_next) .GT. 0) then !haven't been encountered
      lnk(i_next,j_next) = -lnk(i_next,j_next)
      i_next = -lnk(i_next,j_next)
      do 178 j=1,nref
        if (-lnk(i_next,j_next) .EQ. ref(j)) goto 179
178      continue
        goto 177
      endif
179 continue
      h_tim(nj,1)=xj
      plac(nj)=i
200 continue

c  =====
c  Writing to output file in the following sequence :

```

```

c
c      Number of reachable sites, location of each and time to
c      get there;
c      Number of unreachable sites and location of each;
c      Lines connecting each parent-child pair in the path, the
c      lines are defined by the two endpoint locations
c      =====
c      write (3,*) nj
c      do 210 i=1,nj
c      k=plac(i)
c      write (3,122) lat(k)*180./pi,-lon(k)*180./pi
c      & ,bk,int(h_tim(i,1))
122  format (2f12.6,a1,1x,12)
210  continue
c      write (3,*) ncg
c      if (ncg.eq.0) go to 235
c      do 230 i=1,ncg
c      k=loc(i)
c      write (3,*) lat(k)*180./pi,-lon(k)*180./pi
230  continue
235  continue
c      do 220 i=1,nn
c      do 215 j=1,nheli
c      if (i.eq.ref(j)) go to 220
c      if (lnk(i,j).GE.0) go to 215
c      write (3,*) 2
c      write (3,*) lat(-lnk(i,j))*180./pi,-lon(-lnk(i,j))*180./pi
c      write (3,*) lat(i)*180./pi,-lon(i)*180./pi
215  continue
220  continue

c      write(*,*) " ***** End of Processing *****"
c      goto 1001
1000 continue
c      write(*,*)
c      write(*,*) " ***** Process Aborted *****"
1001 continue
c      end

```

```

c =====
c      Function dst
c
c      This function find the distance between two points. The
c      location of these points are defined by their latitudes and
c      longitudes in radian.
c
c      Input : b      pi/2 - latitude of first point
c              c      pi/2 - latitude of second point
c              biga    different in longitude of first and second point
c
c      Return : 0 if error occurs
c              distance otherwise
c =====
c      function dst(b,c,biga,erad)
c      x=cos(b)*cos(c)+sin(b)*sin(c)*cos(biga)
c      if (x.ge.1.) then
c      dst=0.
c      return
c      else
c      dst=acos(x)*erad
c      return
c      endif
c      end

```

```

c =====
c      Function find_rg_sp
c
c      This function find the range and speed of the available
c      helicopter type for a given reference point. The first type
c      found will be returned since we are assuming a one type for
c      a given reference point.
c
c      Input : ref    index of the reference point
c
c      Output : rang   range of helicopter type used
c              sp      speed of helicopter type used
c

```

```

c Return : .TRUE.      >= 1 helicopter type is available
c           .FALSE.    no helicopter type is available
c =====
      function find_rg_sp(ref,rang,sp)
      logical find_rg_sp
      integer ref
      real rang, sp
      real range(10), spd(10)
      integer nheli, heli_ref(10,10)
      common /heli_info/range,spd,nheli,heli_ref,max_ref
      do 100 i=1,nheli
        j = 1
200      continue
        if (heli_ref(i,j) .EQ. 0 .OR. j .EQ. max_ref) goto 100
        if (heli_ref(i,j) .EQ. ref) then
          find_rg_sp = .TRUE.
          rang = range(i)
          sp = spd(i)
          return
        endif
        j = j + 1
        goto 200
100      continue
      find_rg_sp = .FALSE.
      return
      end

```

```

c *****
c
c function name_len
c
c       Returns the length of character string with the right
c hand side spaces being trimmed
c
c *****
      function name_len(a)
      character*(*) a
      do 10 i=len(a),1,-1
        if ( a(i:i) .NE. ' ') then
          name_len = i
          return
        endif
10      continue
      name_len = 0
      return
      end

```

```

c *****
c
c function name_req
c
c       Returns true if a is exactly same as b with the right
c hand side spaces are being ignored.
c
c *****
      function name_req(a,b)
      character*(*) a,b
      ia = name_len(a)
      ib = name_len(b)
      name_req = .false.
      if (a(1:ia) .EQ. b(1:ib)) name_req = .true.
      return
      end

```



```

! backup in : MPMG.tar
c *****
c
c Program Name : sarmap.for
c
c Function : This program plots map data from file g1.src using a
c Calcomp plotter. Informations of different kinds are read
c in from different input files and overlay on the map.
c
c Input : refnam contains transit time and coverage of all refilling
c sites by showing the shortest transit time and the
c path (shortest time path) to get there.
c This file should be generated from base.for
c fnam filename of overlay line to be drawn
c plnm filename of overlay points to be plotted
c
c Output : file named "sarmap.out" will be created. It is a postscript file
c ready to be view using "pageview" or other postscript viewer.
c
c Note : A complete Canada map is contained in g1.src, the region
c being generated is determined by the boundaries given in
c the parameter file (file1 if any) or from stdin.
c
c Date : 26 May 1994
c *****
c *****
c
c Other Variables Used
c
c proj has a value of
c 1 for a Lambert Conformal Projection
c 2 for a Mercator Projection
c res has the value 1 for a fine data base map requiring a tape mount.
c For other res values map data is obtained from the disc file
c wrld.data.
c cont specifies the tape to be mounted when res=1
c 1 for Africa tape UG0391
c 2 for Europe tape UG0392
c 3 for North America tape UG0393
c 4 for South America tape UG0394
c 5 for Asia tape UG0395
c xnl1,slt,wln,eln define the boundaries of the map window to be drawn.
c They are the north latitude, south latitude, west longitude
c and east longitude respectively.
c siz the largest dimension of the plot in inches
c yorn 'y' if an overlay of points is to be plotted, otherwise not
c *****
c character*6 volume_id(5)
c data volume_id/"UG0391","UG0392","UG0393","UG0394","UG0395"/
c character*1 yorn
c character*1 coord
c character*40 title1,title2
c character*256 title
c character*168 file1,file2,fnam,plnm,file_titles,file_panel,
c & file_circle,refnam
c integer proj,cont
c logical overlay,new_dev
c real.lnrf
c data ldev/6/
c data pi/3.1415927/
c data new_dev/.false./
c common / map_constants / radg,pio4,proj
c common / rgbg / i_red,i_green,i_blue,i_gray
c common / textfont / i_size,i_font
c data i_size/8/,i_font/0/
c data i_red/1/,i_green/2/,i_blue/3/,i_gray/4/
c *****
c
c Main Program
c *****
c radg=pi/180.
c pio4=pi/4.
c write(*,*) "Input parameter file name "

```

```

        read(*,900) file1
900 format(a)
        if ( file1 .eq. " " ) then
            i_file = 5
        else
            i_file = 4
            open (4,file = file1 ,form="formatted")
        endif

c *****
c                                     OPEN OUTPUT FILE
c *****
        open (10,file="sarmap.out")

c =====
c If overlay, file2 is supposed to contain the grid of the map
c and we will only plot the information
c =====
        write(*,*) "Input overlay file name "
        read(*,900) file2
        if ( file2 .eq. " " ) then
            overlay = .false.
        else
            overlay = .true.
        endif

c =====
c Read from parameter file, if exists. Otherwise from stdin
c =====
        write(*,*) "Do you have files of input points and lines? "
        read (i_file,*) yorn
        if ( yorn .ne. 'y' ) then
            else
                write(*,*) "Input file name of refuelling points "
                read (i_file,900) refnam
                write(*,*) "Input file name of lines "
                read (i_file,900) fnam
                write(*,*) "Input file name of points "
                read (i_file,900) plnm
                write(*,*) "Input file name of titles "
                read (i_file,900) file_titles
                write(*,*) "Input file name of panels "
                read (i_file,900) file_panel
                write(*,*) "Input file name of circles "
                read (i_file,900) file_circle
            endif
            write(*,*) "Input window (north latitude, south latitude "
            write(*,*) "                west longitude, east longitude "
            read (i_file,*) xnlt,slt,wln,eln
            write(*,*) "Input proj (lambert = 1, mercator=2) "
            read (i_file,*) proj
            write(*,*) "Input resolution (low=1,"
1          " high = 2 (requires tape mount))"
            read (i_file,*) res
            if ( res .ge. 2 ) then
                write(*,*) "Input continent "
                read (i_file,*) cont
            endif
            write(*,*) "Input output plot size (as a % of full size)"
            read (i_file,*) siz
            siz = siz/100.
            if ( siz .le. 0. ) siz = 1.
            if ( siz .gt. 1. ) siz = 1.

            write (*,*) "Input title first line (default => file name)"
            read (i_file,5) title1
            write (*,*) "Input title second line (default => file name)"
            read (i_file,5) title2
5          format (a40)

        10 continue
        call grain(0.1)
c =====
c We can now initialize some constants
c =====
        call map_init(xnlt,slt,wln,eln,siz)

```

```

    if ( proj .eq. 1 ) then
      lnrf = (wln + eln)/2.
      call init_lamb(xnlt,slt,lnrf)
    endif

    call txicur(2,8)
    call aspect
    ires=res+0.5
    if (ires .eq. 1 ) then
c =====
c This version uses permanent common to store the data
c =====
      else
        if ( cont .eq. 3 ) then
c          open (1,form="formatted",mode="in",
c          &      attach="tape_nstd_UG0393 -bk 5076 -den 6250 -tk 9",
c          &      err=50,iostat=ios)
        else
c          open (1,form="formatted",mode="in",
c          &      attach="tape_nstd_ "//volume_id(cont)//
c          &      " -bk 5076 -den 6250 -tk 9",
c          &      err=50,iostat=ios)
        endif
        rewind (1,err=60,iostat=ios)
        if (ios.ne.0) go to 90
      endif

c =====
c We have to draw grid if there is no overlay file, which have
c the grid there. map1 draw grid with low resolution and map2
c draw with high resolution
c =====
      if ( .not. overlay ) then
        call map_grid(xnlt,slt,wln,eln,refnam,title1,title2)
        if ( ires .eq. 1 ) then
          call make_map1(xnlt,slt,wln,eln)
        else if ( ires .eq. 2 ) then
          call make_map2(xnlt,slt,wln,eln,proj,cont)
        endif
      endif

c =====
c We have information to put on top of the map if yorn is 'y'
c =====
      if (yorn.eq. 'y') then
        if ( file_panel .ne. " ") then
          open (97,file=file_panel,form="formatted")
          call map_panels(xnlt,slt,wln,eln,.true.)
          close(97)
        endif
        if ( file_titles .ne. " ") then
20          open (97,file=file_titles,form="formatted")
          read(97,*,end=1020) coord,x,y
          read(97,*,end=1020) title
          call map_title(x,y,coord,title)
          goto 20
1020        continue
          close(97)
        endif
        if ( file_circle .ne. " ") then
          open (97,file=file_circle,form="formatted")
          call map_circles(xnlt,slt,wln,eln,proj)
          close(97)
        endif
        if ( fnam .ne. " ") then
          open (97,file=fnam,form="formatted")
          call map_lines(xnlt,slt,wln,eln,proj)
          close(97)
        endif
        if ( plnm .ne. " ") then
          open (98,file=plnm,form="formatted")
          call map_points(xnlt,slt,wln,eln,proj)
          close(98)
        endif
        if ( refnam .ne. " ") then
          open (97,file=refnam,form="formatted")

```

```

        call map_refill(xnlt,slt,wln,eln)
        close(97)
      endif
    endif

c =====
c There is no where new_dev is modified in the program and
c there will be an infinite loop when it is .TRUE., therefore
c The program MUST be modified if we're using this later
c =====
      if ( new_dev ) goto 10
      call map_end
      stop

c =====
c File error handling
c =====
50   write(*,*) " ERROR ON TAPE OPEN ",ios
1001 write(*,*) "open error on calcomp file "
      stop
55   ioab=iabs(ios)
      stop
60   write(*,*) " ERROR ON TAPE REWIND ",ios
      go to 55
90   write(*,*) " iox find failed ",ios
      go to 55
      end

c *****
c
c   function tanz
c
c *****
      function tanz(lat,e)
      implicit double precision (a-z)
      data pi/3.141592654/
      p=pi/4.+lat/2.
      q1=e*sin(lat)
      q2=(1.-q1)/(1.+q1)
      z=tan(p)*q2**(e/2.)
      tanz=log(z)
      return
      end

c *****
c
c   subroutine make_map1
c
c   This routine put the world map into the postscript file
c
c   Input :      File "g1.src" contains the information of points
c                to be draw to create the map
c
c   Output :     output file fort.10 will contains the converted
c                information from g1.src in postscript form
c
c *****
c   subroutine make_map1(xnlt,slt,wln,eln)

      common / map_constants / radg,pio4,proj
      common / rgbg / i_red,i_green,i_blue,i_gray
      real radg,pio4,lnrf,qlat,qlon
      integer proj,res
      real lat,lon
      data pi / 3.1415927 /
      logical skip
      character*2 name,name2
      character*6 names
      logical name_req
      data names /'CAUSGL'/

      lnrf = (wln + eln)/2.
      call init_lamb(xnlt,slt,lnrf)

      xc = 0.

```

```

      j=1
      nj=0
      skip = .true.
      n_count = 1
      j_count = 1
      open(97,file="g1.src")
      read(97,910) name

      call linclr(1.,i_green)
350 continue

      read(97,910) name,name2
910 format(a2,8x,a2)

      if ( .not. (name_req(name,names) .or.
&          name_req(name2,names)) ) then
360 continue
      read(97,900,end=1000) qlat
      if ( qlat .ne. 999999. ) goto 360
      goto 350.
      endif

370 read(97,900,end=1000) qlat,qlon,res
      if ( qlat .eq. 999999. ) then
          skip = .true.
          goto 350
      endif
900 format(f6.0,1x,f7.0,2x,i1)

c =====
c Include all points with resolution from 1 to 5, points with
c res of 1 is of higher resolution than points with res 5
c Note : if (res .lt. 5) is used, a rough map is formed
c =====
      if ( res .lt. 1 ) goto 370
      j=j+1
      n_count = n_count + 1
      lat=90.-qlat/3600.
      lon=qlon/3600.
      if ( lon .gt. 180. ) lon = lon - 360.
      if ( skip ) then
          call map_move(lat,lon)
          skip = .false.
      else
          call map_draw(lat,lon)
      endif
      goto 370
1000 close(97)
      write(*,*) n_count,j,j_count
      call map_move(lat,lon) !initiate a stroke to finish the path
      call linclr(1.,0)
      return
      end

c *****
c
c subroutine make_map2
c
c *****
      subroutine make_map2(xnlt,slt,wln,eln,proj,cont)
      real qlat(108),qlon(108),lat,lon
      integer seq(108),rank(108),file(108)
      integer proj,cont
      data pi / 3.1415927 /
      logical skip
      radg = pi / 180.
      pio4 = pi / 4.
227 continue ! Plots map from points of tape unit 1 which are inside window.
      npw=0
      nfil=0
      xc=0.
      nsq=0
      ipen=3
      skip = .true.
5      format (108((2f13.6),3(i6,1x)))
10      read (1,5,end=13,err=30) (qlat(m),qlon(m),seq(m),rank(m)

```

```

& ,file(m),m=1,108)
  npw=npw+1
  if (cont.eq.3.and.npw.ge.11191) go to 13 ! Tape UG0393 has bad record
  do 420 i=1,108
    if (seq(i)-nsq) 100,110,100
100  continue
15   format (1x,2f13.6,3i5)
    nsq=seq(i)
    skip = .true.
110  continue
    lat=qlat(i)
    lon=qlon(i)
    if (lat.gt.xnlt.or.lat.lt.slt.or.lon.lt.wln.or.lon.gt.eln) then
      skip = .true.
      goto 420
    endif
    if ( skip ) then
      call map_move(lat,lon)
      skip = .false.
    else
      call map_draw(lat,lon)
    endif
420  continue
    go to 10
13   nfil=nfil+1
2    format (1x,47a1)
    if (nfil.lt.2) go to 10

    return
55   ioab=iabs(ios)
    ios = 1
    stop
30   write(*,*) " ERROR ON TAPE READ ",ios
    write(*,*) " Bad read on tape at record ",npw
    go to 55
80   write(*,*) " iox control failed ",ios
    go to 55
95   write(*,*) " iox mode failed ",ios
    go to 55
end

```

```

c *****
c
c  subroutine map_refill
c
c    This routine draws the refilling sites from input file.
c  The shortest time and path to get to each refilling point is
c  shown. The input file should be generated from the bas6.for
c  or bas9.for which generate the time and path as output.
c
c  Input :      file unit 97, which should be the data file
c               generated from the IFR and VFR analysis from
c               bas6 and bas9 respectively
c
c  Output :     time to reach each individual site is located
c               at the location of site on the map in blue
c               path to reach that site from the nearest base
c               is shown with blue lines and they are the path
c               producing the minimal time shown at the site
c *****
c  subroutine map_refill(xnlt,slt,wln,eln)
c
c    logical skip
c
c    real lat,lon,lnrf
c    data r_colour/1./,i_pat/0/
c    common / rgbg / i_red,i_green,i_blue,i_gray
c
c    lnrf = (wln + eln)/2.
c =====
c  Reads and plots points from unit 97.
c =====
c    read(97,*,end=500) iheli
c    write(*,*) "No. of helicopter types : ",iheli

```

```

do 200 i=1,iheli
  read(97,*,end=500) isp
  read(97,*,end=500) tps
  read(97,*,end=500) irg
  write(*,*) "Helicopter",i
  write(*,*) "Speed      : ",isp
  write(*,*) "Time Per Stop : ",tps
  write(*,*) "Range       : ",irg
200 continue
  call init_lamb(xnlt,slt,lnrf)
c    call lnwidth(1)

c =====
c Reads reachabel and unreachabel site from unit 97.
c =====
do 101 j=1,2
  read(97,*,end=500) npt
  write(*,*) "No. of site : ",npt
  if (j .EQ. 1) then
    call lincir(r_colour,i_blue)
    do 203 i=1,npt
      read (97,*,end=500) lat,lon,i_time
      if (lat.gt.xnlt.or.lat.lt.slt.or.
&        lon.lt.wln.or.lon.gt.eln) then
        else
          call map_number(lat,lon,i_time)
        endif
203    continue
      call lincir(1.,0)
    else
      do 204 i=1,npt
        read (97,*,end=500) lat,lon
        if (lat.gt.xnlt.or.lat.lt.slt.or.
&          lon.lt.wln.or.lon.gt.eln) then
          else
            call map_marker(lat,lon,i_symbol,i_colour)
          endif
204      continue
    endif
101 continue

    call init_lamb(xnlt,slt,lnrf)
c =====
c Reads and draw path unit 97.
c The number of points, npt in a path is read and the locations
c of these npt points are read and drawn as connected points
c =====
344 continue
  read(97,*,end=500) npt
  skip = .true.
  do 321 i=1,npt
    read (97,*,end=500) lat,lon
    if (i.eq.1) skip = .true.
    if (lat.gt.xnlt.or.lat.lt.slt.or.lon.lt.wln.or.lon.gt.eln) then
      skip = .true.
    else
      if ( skip ) then
        call map_move(lat,lon)
        skip = .false.
      else
        call map_draw(lat,lon)
      endif
    endif
321 continue
  go to 344
500 continue
  call map_move(lat,lon)
c    call lnwidth(0)
  return
end

c *****
c
c subroutine map_lines
c

```

```

c *****
  subroutine map_lines(xnlt,slt,wln,eln,proj)

    integer proj
    logical skip

    real lat,lon,lnrf
    data pi / 3.1415927 /

    radg = pi / 180.
    pio4 = pi / 4.
    lnrf = (wln + eln)/2.
c =====
c Reads and plots points from unit 97.
c =====

345 read (97,900,end=225) npt, i_colour,i_pat
900 format(i5,i5,i5)
    skip = .true.
    call dashpt(i_pat)
    call linclr(REAL(i_colour),0)
    xc=0.
    call init_lamb(xnlt,slt,lnrf)
    do 320 i=1,npt
      read (97,*,end=225) lat,lon
      if (i.eq.1) skip = .true.
      if (lat.gt.xnlt.or.lat.lt.slt.or.lon.lt.wln.or.lon.gt.eln) then
        skip = .true.
      else
        if ( skip ) then
          call map_move(lat,lon)
          skip = .false.
        else
          call map_draw(lat,lon)
        endif
      endif
    320 continue
    go to 345
225 call linclr(1.,0)
    call dashpt(0)
    return
  end

c *****
c
c  subroutine map_points
c
c    This routine plot points on map using the desired marked
c
c *****
  subroutine map_points(xnlt,slt,wln,eln,proj)

    integer proj

    real lat,lon,lnrf
    data pi / 3.1415927 /

    radg = pi / 180.
    pio4 = pi / 4.
    lnrf = (wln + eln)/2.

    10 read (98,900,end=1000) npt,i_symbol,i_colour
900 format(i5,i5,i5)

    do 202 i=1,npt
      read (98,*,end=1000) lat,lon
      if (lat.gt.xnlt.or.lat.lt.slt.or.lon.lt.wln.or.lon.gt.eln) then
        else
          call map_marker(lat,lon,i_symbol,i_colour)
        endif
    202 continue
    goto 10

1000 continue
    call linclr(1.,0)
    call dashpt(0)
    return

```


end

```

c *****
c
c  subroutine init_lamb
c
c *****
c      subroutine init_lamb(lat1,lat2,lnrf)
c  Computes Lambert Conformal coordinates x,y for lat and lon
c      implicit double precision(a-z)
c      real lat1,lat2,lat,lon,xs,ys,lnrf,r,theta,dist
c      real * 8 x
c      data pi/3.141592654/
c      save

c      rho(x,a,ee)=a/dsqrt(1.-ee*dsin(x)*dsin(x))

c      lat_1 = lat1 * 2./3. + lat2 * 1./3.
c      lat_2 = lat1 * 1./3. + lat2 * 2./3.

c      a=6378206.4d0
c      b=6356583.8d0
c      ee=1.d0-(b*b)/(a*a)
c      e=sqrt(ee)
c      c=pi/180.d0
c      lat_1=lat_1*c
c      lat_2=lat_2*c
c      lnrf_1=lnrf*c
c      z1=tanz(lat_1,e)
c      z2=tanz(lat_2,e)
c      n1=rho(lat_1,a,ee)
c      n2=rho(lat_2,a,ee)
c      xl=dlog(n1/n2*dcos(lat_1)/dcos(lat_2))/(z2-z1)
c      xk=n1*dcos(lat_1)/(xl*exp(-xl*z1))
c      theta0=dasin(xl)
c      r0=rho(theta0,a,ee)/dtan(theta0)
c      write(*,*) " xl,xk,theta0,r0 ",xl,xk,theta0,r0
c  ... end of initial constants
c      call pivot(0.,r0*scal)
c      return

c =====
c  entry lamb2
c =====
c      entry lamb2(lat,lon,xs,ys,dist,theta)
c      gamma=(lon*c-lnrf_1)*xl
c      xlat=lat*c
c      r=xk*exp(-xl*tanz(xlat,e))
c      xs=r*dsin(gamma)
c      ys=r0-r*dcos(gamma)
c      dist = r
c      theta = -pi/2 + gamma
c      return
c  end !init_lamb

c *****
c
c  subroutine map_move
c
c      This routine update the current cursor position on the
c  map in the postscript file. The actual location of point in
c  terms of latitude and longitude is passed, together with the
c  type of projection (1 and 2 for Lambert and Mercator resp.)
c  we are using, the coordinate of the passed point in world
c  coordinate is found.
c
c      In map_move, a stroke is called before moving to the
c  next cursor position to clear the previous path drawn.
c
c      In map_draw, a line is draw from the current position
c  to the coordinate being passed and the current position is
c  updated to be the passed position. The path created but not
c  drawn yet before a stroke is called.
c
c      In map_number, we are putting an number in the specified

```

c after moving to that location. The number is to be passed in
 c as an integer. Must be a number less than 3 digits (2 digits
 c for negative numbers), modification on subroutine number will
 c allows a larger number with more digits.

c
 c In map_marker, we are putting a marker in the specified
 c after moving to that location. The marker to be put on the
 c map is determined by the marker id being passed.

c *****

```
subroutine map_move(lat,lon)
  real lat,lon,x,y,r,theta
  integer proj
  data pi/3.1415927/
  common / map_constants / radg,pio4,proj
```

```
  if ( proj .eq. 1 ) then
    call lamb2(lat,lon,x,y,r,theta)
  c    call mpolar(r*scal,theta)
    call mpolar(r,theta)
  else
    x=lon*radg
    y=alog(tan(pio4+(lat*radg/2.)))
    call stroke
    call move(x,y)
  endif
  return
```

```
c =====
c entry map_draw
c =====
  entry map_draw(lat,lon)
  if ( proj .eq. 1 ) then
    call lamb2(lat,lon,x,y,r,theta)
    call dpolar(r,theta)
  c    call dpolar(r*scal,theta)
  else
    x=lon*radg
    y=alog(tan(pio4+(lat*radg/2.)))
    call draw(x,y)
  endif
  return
```

```
c =====
c entry map_number
c
c Write a integer on map at specifed location
c =====
  entry map_number(lat,lon,i)
  if ( proj .eq. 1 ) then
    call lamb2(lat,lon,x,y,r,theta)
    call mpolar(r,theta)
  c    call mpolar(r*scal,theta)
  else
    x=lon*radg
    y=alog(tan(pio4+(lat*radg/2.)))
    call move(x,y)
  endif
  call number(i)
  return
```

```
c =====
c entry map_marker
c =====
  entry map_marker(lat,lon,i_symbol,i_colour)

  call mrkclr(i_colour)
  if ( proj .eq. 1 ) then
    call lamb2(lat,lon,x,y,r,theta)
  else
    x=lon*radg
    y=alog(tan(pio4+(lat*radg/2.)))
  endif
  call marker(x,y,i_symbol)
```

```

return
end !map_move

```

```

c *****
c
c  subroutine map_init
c
c      Initializing the variables determining how the map is
c      to be scaled and drawn.
c
c  Variable being initialized :
c
c      xscale, yscale - the scaling factor in x and y direction
c                      they determine the size of map
c      xtrans, ytrans - the translation factor in x,y direction
c                      they determine where the map is located
c                      in the paper
c
c      The above variables are determined by the boundaries of
c      map being drawn, i.e. the two latitudes and two longitudes, as
c      well as the output plot size
c
c  Input :      xnlt, slt, wln, eln      boundaries
c              siz      output plot size (in %)
c
c  Output :     xscale, yscale, xtrans, ytrans being initialized
c
c *****
c      subroutine map_init(xnlt,slt,wln,eln,siz)
c      data pi/3.1415927/
c      common / map_constants / radg,pio4,proj
c      real xscale, yscale, xtrans, ytrans
c      common /scale_vars/ xscale, yscale, xtrans, ytrans
c      common /txpos/ x_shift
c      integer proj
c      real lnrf
c      xscale = 1.
c      yscale = 1.
c      xtrans = 72.
c      ytrans = 72.
c      xtrans = 60.
c      ytrans = 130.
c      hlf = 50.
c      lnrf = ( wln + eln ) / 2.
c      call init_postscript
c      scal = 1.0
c      if ( proj .eq. 1 ) then
c          call init_lamb(xnlt,slt,lnrf)
c          call lamb2(xnlt,eln,x_max,y_max,r_top,theta_top)
c          call lamb2(slt,lnrf,x_min,y_min,r_bot,theta_bot)
c          call lamb2(slt,eln,x_max,ye,re,thetae)
c          if ( ye .gt. y_max ) then
c              y_max = ye
c              x_max = r_bot
c          endif
c          x_min = - x_max
c      else
c          y_max = alog( tan( pio4 + ( xnlt * radg )/2. ))
c          y_min = alog( tan( pio4 + ( slt * radg )/2. ))
c          x_max = eln * radg
c          x_min = wln * radg
c      endif
c
c      call ndc2w2(100.,100.,x,y)
c      if ( siz .ne. 1. ) then
c          call window(x*((1.-siz)/2.),x*(1.-(1.-siz)/2.),
c          &          y*((1.-siz)/2.),y*(1.-(1.-siz)/2.))
c          call viewport(x*((1.-siz)/2.),x*(1.-(1.-siz)/2.),
c          &          y*((1.-siz)/2.),y*(1.-(1.-siz)/2.))
c      endif
c
c      scal_x = siz*x/(x_max-x_min)
c      scal_y = siz*y/(y_max-y_min)
c      scl = min(scal_x,scal_y)

```

```

      call scale(scl,scl)
      call scale(.8,.8)
      call transl(-x_min/siz,-y_min/siz)

c      call ndc2w2(100.,100.,x,y)
c      if ( x .gt. x_max ) then
c          call transl((x-x_max)/2.,0.)
c      endif
c      if ( y .gt. y_max ) then
c          call transl(0.,(y-y_max)/2.)
c      endif

      call pivot(0.,(y_min+r_bot))
      call linclr(1.,0)
      return
      end

c *****
c
c      subroutine map_end
c
c          After this routines is called, the following tasks are
c          being perform :
c
c          - all the file should be closed
c          - any dynamic variables being allocated are free
c          - the postscript file containing the map is ended with
c            appropriate postscript commands
c
c *****
c      subroutine map_end
c      call end_postscript
c      return
c      end

c *****
c
c      subroutine map_grid
c
c          Prepare a clipped region for the map, as well as putting
c          in the latitudes, longitudes and labels on the map.
c
c      Input :      xnlt, slt, wln, eln are the four boundaries of the
c                  region being clipped in terms of latitude(S and N)
c                  and longitude(E and W)
c
c                  refnam is the data file containing the shortest time
c                  and path to reach each refilling sites
c
c      Output :      The grid of map is defined, it is being used as the
c                  clipping region for the map content.
c
c                  Two latitudes and two longitudes within the clipped
c                  region are found and being drawn, labelled.
c
c                  The filename of refilling data are being labelled
c                  at the bottom of the map indicating the source
c                  of data (the refilling sites) to be plotted.
c
c *****
c      subroutine map_grid(xnlt,slt,wln,eln,refnam,title1,title2)
c      data pi/3.1415927/
c      data lsize/12/
c      common / map_constants / radg,pio4,proj
c      integer proj
c      real lat1,lat2,long1,long2
c      character*(*) refnam,title1,title2
c      character*168 flabel
c      character * 7 label

      lat1 = (2.*xnlt + slt)/3.
      lat2 = ( xnlt + 2.*slt)/3.
      lat_s = (xnlt - slt)/10.
      long1 = (2.*wln + eln)/3.

```

```

long2 = ( wln + 2.0*eln)/3.
long_s= (wln - eln)/10.

c =====
c We first put in some labels for latitudes
c =====
call txsize(lsize)

c call map_latlon(lat1,"SN",label)
c call map_label(lat1,wln,6,label,lsize)
c call map_label(lat1,eln,4,label,lsize)

c call map_latlon(lat2,"SN",label)
c call map_label(lat2,wln,6,label,lsize)
c call map_label(lat2,eln,4,label,lsize)

c call map_latlon(long1,"WE",label)
c call map_label(slt-abs((slt-xnlt)/15.),long1,8,label,lsize)
c call map_label(slt-abs((slt-xnlt)/15.),long1,2,label,lsize)
c if ( proj .eq. 2 ) call map_label(xnlt,long1,2,label,lsize)

c call map_latlon(long2,"WE",label)
c call map_label(slt-abs((slt-xnlt)/15.),long2,8,label,lsize)
c call map_label(slt-abs((slt-xnlt)/15.),long2,2,label,lsize)
c if ( proj .eq. 2 ) call map_label(xnlt,long2,2,label,lsize)

c =====
c the following segment of code is actually the call the
c map_label except it is move one line futher down the horizon
c =====
call map_move(slt-abs((slt-xnlt)/15.), (wln+eln)/2.2)
call txicur(8,lsize)
call txicur(8,lsize)
call txfont(11)

c =====
c we now make title out of the data file
c =====

c use a free formatted title if one was specified

i_len = name_len(title1)+name_len(title2)
if (i_len .GT. 1) then
  i_len=name_len(title1)
  call map_move(slt-abs((slt-xnlt)/15.), (wln+eln)/2.1)
  if (i_len .gt. 0) call text(i_len,title1(1:i_len))
  call map_move(slt-abs((slt-xnlt)/15.), (wln+eln)/2.1)
  call txicur(8,lsize)
  call txicur(8,lsize)
  i_len=name_len(title2)
  if (i_len .GT. 0) call text(i_len,title2(1:i_len))

c if no title specified, use the default file name

else
  i_len = name_len(refnam)
  flabel(1:i_len) = refnam(1:i_len)
  do 100 i=1,i_len
    if (flabel(i:i) .EQ. '.') flabel(i:i) = ' '
100 continue
  call text(i_len+LEN(" TRANSIT TIME AND COVERAGE"),
    & flabel(1:i_len)//" TRANSIT TIME AND COVERAGE")

! call text(LEN("Transit Hours and Locations with")+3,
! & "Transit Hours and Locations //"refnam(1:3)//" with")
! call map_move(slt-abs((slt-xnlt)/15.), (wln+eln)/2.1)
! call txicur(8,lsize)
! call txicur(8,lsize)
! call txicur(8,lsize)
! if (name_len(refnam) .GE. 11) then
! call text(LEN(" NM Hop knots from MOBs")+6,refnam(5:7)//
! & " NM Hop //"refnam(9:11)//" knots from MOBs")
! else if (name_len(refnam) .GE. 9) then
! call text(LEN(" NM Hop knots from MOBs")+6,refnam(4:6)//
! & " NM Hop //"refnam(7:9)//" knots from MOBs")
! endif
call txfont(0)

```

```

      call txsize(8)

c =====
c  we now draw outline of area
c =====
      call map_move(xnlt,wln)
      call map_draw(xnlt,eln)
      call map_draw(slt,eln)
      call map_draw(slt,wln)
      call map_draw(xnlt,wln)

c =====
c  set clip region
c =====
      call map_move(xnlt,wln)
      call map_draw(xnlt,eln)
      call map_draw(slt,eln)
      call map_draw(slt,wln)
      call map_draw(xnlt,wln)
      call clip

c =====
c  We now draw in two latitude lines
c =====
      call map_move(lat1,wln)
      call map_draw(lat1,eln)
      call map_move(lat2,eln)
      call map_draw(lat2,wln)

c =====
c  We now draw in two longitude lines
c =====
      call map_move(xnlt,long1)
      call map_draw(slt,long1)
      call map_move(slt,long2)
      call map_draw(xnlt,long2)
      call map_move(slt,long2) !to put the stroke on
      return
      end

c *****
c
c  subroutine map_label
c
c      Put a text string of specified lenght on a specified
c  position relative to given initial position.
c *****
      subroutine map_label(x,y,ipos,text_out,text_size)
      real x,y
      character*(*) text_out
      integer text_size
      call map_move(x,y)
      call txcur(ipos,text_size)
      call text(-1,text_out) !length of text_out is not known
      return
      end

c *****
c
c  subroutine map_latlon
c
c      Create label given the latitude and longitude
c *****
      subroutine map_latlon(x,ew,text_out)
      character*(*) text_out
      character*1 ew1
      character*2 ew
      if ( x .lt. 0. ) then
         ew1 = ew(1:1)
         y = -x
      else
         ew1 = ew(2:2)

```

```

      y = x
    endif
    call cvr2c(y,6,text_out,2)
    text_out = text_out(1:6)//ew1
    return
  end

```

```

c *****
c
c subroutine map_title
c
c *****
  subroutine map_title(x,y,coord,text_out)
    real x,y,xw,yw
    character*(*) text_out
    character*1 coord
    call txfont(0)
    call txqual(4)
    call txsize(0)
    if ( coord .eq. "W" .or. coord .eq. "w" ) then
      call map_move(x,y)
    else if ( coord .eq. "S" .or. coord .eq. "s" ) then
      call ndc2w2(x,y,xw,yw)
      call move(xw,yw)
    else if ( coord .eq. "L" .or. coord .eq. "l" ) then
      call ndc2w2(x,y,xw,yw)
      call move(xw,yw)
    endif
    call margin(0.0)
    do 10 i=len(text_out),1,-1
      if ( text_out(i:i).ne. " " ) then
        call typset(-1,"!LIN-0.5;"/>!LIN!END")
        goto 20
      endif
    10 continue
    20 continue
    if ( coord .eq. "L" .or. coord .eq. "l" ) then
      call txfont(0)
      call txsize(0)
      call map_panels(xnlt,slt,wln,eln,.false.)
    endif
    return
  end

```

```

c *****
c
c subroutine igl_file_init
c
c *****
  subroutine igl_file_init

    inquire(file="IGLFNT",exist=exists)
    if ( .not. exists ) then
      call link(">l3p>IGL.07/07/88>d>IGLFNT","IGLFNT")
    endif

    inquire(file="ERRFIL",exist=exists)
    if ( .not. exists ) then
      call link(">l3p>IGL.07/07/88>d>ERRFIL","ERRFIL")
    endif

    do 30 ifnt=1,16
      ifont=ifnt
      call rdfont('IGLFNT',2500,ifont-1,ifont,.true.)
    30 continue
    close(1)
    return
  end

```

```

c *****
c
c subroutine map_panels
c
c *****

```

```

      subroutine map_panels(xnlt,slt,wln,eln,all)

      data pi/3.1415927/
      common / map_constants / radg,pio4,proj
      integer proj
      logical all
      real lat(5000),lon(5000),lnrf,r

      lnrf = (wln + eln)/2.
c =====
c   Reads and plots points from unit 97.
c =====
      10 read (97,900,end=225) npt, i_colour,i_pat
      900 format(i5,i5,i5)

      call linc1r(REAL(i_colour),0)
      xc=0.
      continue
      j = 1
      i = 1
      n_pt = 1
      20 read(97,*,end=225) lat(j),lon(j)
         if (.not. all .or. lat(j) .gt. xnlt .or.
           &   lat(j) .lt. slt .or. lon(j) .lt. wln .or.
           &   lon(j) .gt. eln ) then
             else
               n_pt = j
               j = j + 1
             endif
             i = i + 1
             if ( i .le. npt ) goto 20
             if ( n_pt .le. 2 ) goto 1000
             do 100 i = 1, n_pt
               if ( .not. all ) then
                 call ndc2w2(lat(i),lon(i),x,y)
               else if ( proj .eq. 1 ) then
                 call lamb2(lat(i),lon(i),x,y,r,t)
               else
                 x=lon(i)*radg
                 y=log(tan(pio4+(lat(i)*radg/2.)))
               endif
               lat(i)=x
               lon(i)=y
             100 continue

             call filpan(i_pat,.true.)
             call panel(n_pt,lat,lon)

      1000 if ( all ) goto 10
      225 continue
      return
      end

c *****
c
c   subroutine map_circles
c *****
c   subroutine map_circles(xnlt,slt,wln,eln,proj)

      integer proj,colour,fill
      real lat,lon,r
      real xnlt,slt,wln,eln
      real lat1(361), lon1(361)
      data pi / 3.1415927 /

      pio4 = pi / 4.
      radg = pi / 180.
      10 read(97,*,end = 1000 ) lat,lon,r,colour,fill
         lat = lat * radg
         lon = lon * radg
         r = (r/60.)*radg
         cbg = 0.
         xj = 0.
         ov = 0.

```



```

ic = 0
if ( r .gt. ( pi/2.-lat)*60./radg ) ic = 1
do 100 j=1,361
  asm=acos(cos(r)*cos(pi/2.-lat)+sin(r)*sin(pi/2-lat)*
    &      cos(cbg*radg))
  bgb=asin((sin(r)*sin(cbg*radg))/sin(asm))
  lat1(j)=(pi/2.-asm)/radg
  lon1(j)=(lon+bgb)/radg
  if ( ic .eq. 1 .and. bgb .ge. ov ) lon1(j) = ( pi+lon-bgb ) / radg
  cbg = cbg+1.
  ov = bgb

  lat1(j) = max( min( xnit , lat1(j) ) , slt)
  lon1(j) = min( max( wln , lon1(j) ) , eln)

  if ( proj .eq. 1 ) then
    call lamb2(lat1(j),lon1(j),x,y,r1,t)
  else
    x=lon1(j)*radg
    y=alog(tan(pio4+(lat1(j)*radg/2.)))
  endif
  lat1(j)=x
  lon1(j)=y

100 continue

  call lincir(REAL(colour),0)
  if ( fill .lt. 99999 ) then
    call filpan(fill,.false.)
    call panel(200,lat1,lon1)
    call panel(162,lat1(200),lon1(200))
  endif

c   call skip
c   call opnpol
  call poly(200,lat1,lon1)
  call poly(162,lat1(200),lon1(200))
  goto 10

1000 continue
  return
end

c *****
c
c   subroutine init_postscript
c   *****
c     subroutine init_postscript
c       write(10,100) "%!"
c     100 format(a2)
c       call txsize(0)
c     call lnwidth(0)
c       return
c       end

c *****
c
c   subroutine end_postscript
c   *****
c     subroutine end_postscript
c       write(10,*) "stroke showpage"
c       return
c       end

c *****
c
c   subroutine mpolar
c   *****

```

```

subroutine mpolar(r,theta)
real r, theta
return
end

```

```

c *****
c
c subroutine txicur
c
c   It determines the text output position following it
c   before next txicur is called.
c
c   The corresponding movement depends on the font size
c   currently being used.
c
c   Input : ipos ranges from 1 to 9, representing location of
c           text thereafter :
c           1      no change
c           2      centered current horizon
c           3      left justified on current horizon
c           4      move down half line
c           5      centered and move down half line
c           6      left justified and down half line
c           7      move down one line
c           8      centered and down one line
c           9      left justified and down one line
c
c   Output : changed the current pointer y position and set
c            the x-offset for printing character string after
c
c *****
c   subroutine txicur(ipos,fontsize)
c   integer ipos, fontsize
c   real xscale, yscale, xtrans, ytrans
c   common /scale_vars/ xscale, yscale, xtrans, ytrans
c   common /txpos/ x_shift
c
c   if (ipos .LT. 1 .OR. ipos .GT. 9) goto 100
c   if (ipos .GT. 6) then
c     call rmove(0.,-REAL(fontsize)/yscale)
c   elseif (ipos .GT. 3) then
c     call rmove(0.,-REAL(fontsize)/yscale/2.)
c   endif
c   if (MOD(ipos,3) .EQ. 0) then
c     x_shift = -REAL(fontsize*2/3)/xscale
c   elseif (MOD(ipos+1,3) .EQ. 0) then
c     x_shift = -REAL(fontsize*2/3)/xscale/2.
c   else
c     x_shift = 0.
c   endif
c 100 continue
c   return
c   end
c
c *****
c
c subroutine text
c
c   Write text at the specified position set by txicur
c   (left-justified, centered, right-justified) according to
c   the current cursor position into postscript file.
c
c   Input : i      the number of text being displayed
c           text1   text being displayed
c
c   Output : text1 of length i being centered or justified to
c            the pre-set position in a postscript file
c
c *****
c   subroutine text(i,text1)
c   integer i
c   character*(*) text1
c   common /txpos/ x_shift
c
c   length = i
c   if (i .LT. 0) length = LEN(text1)

```

```

c =====
c   Set position of text, x_shift is set in txicur
c =====
c   call rmove(REAL(length)*x_shift,0.)

      write(10,100) text1
100 format ("(",a<length>,") show")
      write(*,100) text1
      return
      end

c *****
c
c   subroutine number
c
c       Write a integer into the postscript file in text
c   form. Length of text being the number of digit the number
c
c *****
      subroutine number(i)
        integer i,c
        real temp
        temp = 1. * i
        c = 1
200 continue
        temp = temp / 10.
        if (temp .GE. 1.) then
          c = c + 1
          goto 200
        endif
        write(10,100) i
100 format ("(",i<c>,") show")
        return
        end

c *****
c
c   subroutine dashpt
c
c       Specifies pattern for dashed lines. Default is solid
c   line, it is used when input number not within range 1 - 4
c
c *****
      subroutine dashpt(i)
        integer i
        if (i .EQ. 1) then
          write(10,*) "[3] 0 setdash"
        elseif (i .EQ. 2) then
          write(10,*) "[3 5] 6 setdash"
        elseif (i .EQ. 3) then
          write(10,*) "[2] 1 setdash"
        elseif (i .EQ. 4) then
          write(10,*) "[2 1] 0 setdash"
        else
          write(10,*) "[ ] 0 setdash"
        endif
        return
        end

c *****
c
c   subroutine lnwidth
c
c       The smaller the number, the thinner the line. 0 is
c   the smallest possible value and it is the default.
c
c *****
      subroutine lnwidth(i)
        integer i
        if (i .GE. 0) then
          write(10,*) i," setlinewidth"
        else
          write(10,*) 2," setlinewidth"
        endif
        return
        end

```

```

subroutine grain(a)
real a
return
end

subroutine poly(i,x,y)
integer i
real x(i),y(i)
return
end

subroutine mrkclr(i)
integer i
return
end

subroutine panel(i,x,y)
integer i
real x(i),y(i)
return
end

subroutine pivot(x,y)
real x,y
return
end

c *****
c
c subroutine linclr
c
c       This routine reset the color used for line drawing.
c
c input :      i      new color intensity
c              Range from 0 to 1 where 0 is the min.
c              intensity value and 1 is the max
c              color   Color id determining the color to be
c              set, as follows :
c              i_red   intensity of red is i
c              i_green intensity of green is i
c              i_blue  intensity of blue is i
c              i_gray  intensity of all three is i
c *****
c subroutine linclr(r,color)
c
c integer color
c real r
c common / rgbg / i_red,i_green,i_blue,i_gray
c
c if (color .EQ. i_red) then
c   write(10,*) r,0,0," setrgbcolor"
c else if (color .EQ. i_green) then
c   write(10,*) 0,r,0," setrgbcolor"
c else if (color .EQ. i_blue) then
c   write(10,*) 0,0,r," setrgbcolor"
c else if (color .EQ. i_gray) then
c   write(10,*) r," setgray"
c else
c   write(10,*) 0," setgray"
c endif
c return
c end

c *****
c
c subroutine marker
c
c *****
c subroutine marker(x,y,i)
c integer i
c real x,y
c call move(x,y-.01)
c call draw(x,y+.01)
c call move(x-.01,y)
c call draw(x+.01,y)

```

```

    call move(x-.01,y-.01)
    call draw(x+.01,y+.01)
    call move(x+.01,y-.01)
    call draw(x-.01,y+.01)
    call stroke
    return
end

```

```

c *****
c
c subroutine txsize
c
c      This routine set the size of character text. It calls
c up txfont to do the postscript file writing.
c
c input :      pts      number of points per character
c                ranged from 4 to 100, default is 8
c
c output :      i_size  set to new font size
c                if pts is out of range, use default
c *****
c      subroutine txsize(pts)
c      integer pts
c      common / textfont / i_size,i_font
c
c      if (pts .GT. 4 .AND. pts .LT. 100) then
c        i_size = pts
c      else
c        i_size = 8
c      endif
c      call txfont(-1)
c      return
c      end
c
c      subroutine dpolar(r,t)
c      real r,t
c      return
c      end
c
c      subroutine aspect
c      return
c      end
c
c      subroutine filpan(i,l)
c      integer i
c      logical l
c      return
c      end
c
c      subroutine margin(a)
c      real a
c      return
c      end

```

```

c *****
c
c subroutine txfont
c
c      This routine select a character font given the font
c number. It is also used to reset the font size after size
c is changed, which is indicated by passing a -1
c
c      New font with i_size defining the size of font will
c be reset in the postscript file.
c
c Input      i : font number, predefined as follows
c
c           0  Courier, defaulted
c           1  Courier-Bold
c           2  Courier-Oblique
c           3  Courier-BoldOblique
c           4  Times-Roman
c           5  Times-Bold
c           6  Times-Oblique
c           7  Times-BoldOblique

```

```

c          8  Helvetica
c          9  Helvetica-Bold
c         10  Helvetica-Oblique
c         11  Helvetica-BoldOblique
c         -1  i_font is used as the current font
c
c          otherwise use default
c
c Output      i_font  set to new font number except when
c              i is -1
c
c *****
c  subroutine txfont(i)
c    integer i
c    character*10 tfont
c    character*30 font
c    common / textfont / i_size,i_font
c
c    if (i .GE. 0) i_font = i
c
c    if (i_font .LE. 3) then
c      tfont = "/Courier"
c      idx = 8
c    elseif (i_font .LE. 7) then
c      if (i_font .EQ. 4) then
c        font = "/Times-Roman"
c        goto 100
c      else
c        tfont = "/Times"
c        idx = 6
c      endif
c    elseif (i_font .LE. 11) then
c      tfont = "/Helvetica"
c      idx = 10
c    else
c      tfont = "/Courier"
c      idx = 8
c      i_font = 0
c      goto 100
c    endif
c
c    if (MOD(i_font,4) .EQ. 0) then
c      font = tfont
c      goto 100
c    endif
c    if (MOD(i_font,4) .EQ. 1) font = tfont(1:idx)//"-Bold"
c    if (MOD(i_font,4) .EQ. 2) font = tfont(1:idx)//"-Oblique"
c    if (MOD(i_font,4) .EQ. 3) font = tfont(1:idx)//"-BoldOblique"
c
c 100 continue
c    write(10,*) font," findfont"
c    write(10,*) i_size," scalefont"
c    write(10,*) "setfont"
c    return
c    end
c *****
c  subroutine txqual
c *****
c    subroutine txqual(i)
c      integer i
c      return
c    end
c *****
c  subroutine typset
c *****
c    subroutine typset(i,text)
c      character*(*) text
c      integer i
c      return
c    end
c *****
c  subroutine ndc2w2
c *****
c    subroutine ndc2w2(x,y,z,w)

```

```

      real x,y,z,w
      z=8.5*72.*100./x
      w=11.*72.*100./y
      return
      end

```

```

c *****
c
c  subroutine cvr2c
c
c      Convert real number to numerical character strings.
c
c  Input : a      the real number to be converted
c          i      length of text
c          j      number of digits to appear past the decimal
c                point, it must be range from 0 - 23 inclusive
c
c  Output : text  result of conversion, in string format
c                filled with '*' if string not long enough
c *****
c  subroutine cvr2c(a,i,text,j)
c      integer i,j
c      real a
c      character*(*) text
c      logical neg
c
c      if (i .LE. 0) goto 1000
c      if (j .LT. 0 .OR. j .GE. 23) goto 1000
c      if (j .GT. i-1) goto 100
c      i_a = INT(a * (10.**j))
c      if (i_a .LE. 0) then
c          neg = .TRUE.
c          i_a = -i_a
c      else
c          neg = .FALSE.
c      endif
c      300 do 400 k=0,j-1
c          text(i-k:i-k) = CHAR(48+i_a-i_a/10*10)
c          i_a = i_a / 10
c      400 continue
c          text(i-j:i-j) = ','
c
c      500 do 600 k=i-j-1,1,-1
c          if (ABS(i_a) .GT. 0) then
c              text(k:k) = CHAR(48+i_a-i_a/10*10)
c              i_a = i_a / 10
c          elseif (neg) then
c              text(k:k) = '-'
c              neg = .FALSE.
c          else
c              text(k:k) = ' '
c          endif
c      600 continue
c          if (.NOT. neg) goto 1000
c
c      100 continue
c          do 200 k=1,i
c              text(k:k) = '*'
c          200 continue
c      1000 continue
c          return
c          end

```

```

c *****
c
c  subroutine cvi2c
c
c      Convert integer number to numerical character strings.
c
c  Input : a      the integer number to be converted
c          i      length of text
c
c  Output : text  result of conversion, in string format
c

```

```

c Note : if string not long enough, then only the right hand
c side digits are stored in the text
c
c *****
c subroutine cvi2c(a,i,text)
c   integer i
c   real a
c   character*(*) text
c   logical neg
c
c   if (i .LE. 0) goto 1000
c   if (i_a .LE. 0) then
c     neg = .TRUE.
c     i_a = -a
c   else
c     neg = .FALSE.
c     i_a = a
c   endif
c
c 500 do 600 k=i,1,-1
c   if (ABS(i_a) .GT. 0) then
c     text(k:k) = CHAR(48+i_a-i_a/10*10)
c     i_a = i_a / 10
c   elseif (neg) then
c     text(k:k) = '-'
c     neg = .FALSE.
c   else
c     text(k:k) = ' '
c   endif
c 600 continue
c   if (.NOT. neg) goto 1000
c
c 100 continue
c   do 200 k=1,i
c     text(k:k) = '*'
c 200 continue
c 1000 continue
c   return
c   end
c
c *****
c
c subroutine scale
c
c *****
c subroutine scale(x,y)
c   real x,y
c   real xscale, yscale, xtrans, ytrans
c   common /scale_vars/ xscale, yscale, xtrans, ytrans
c   xscale = xscale * x
c   yscale = yscale * y
c   xtrans = xtrans / x
c   ytrans = ytrans / y
c   return
c   end
c
c *****
c
c subroutine transl
c
c *****
c subroutine transl(x,y)
c   real x,y
c   real xscale, yscale, xtrans, ytrans
c   common /scale_vars/ xscale, yscale, xtrans, ytrans
c   xtrans = xtrans + x
c   ytrans = ytrans + y
c   return
c   end
c
c *****
c
c subroutine stroke
c
c   Tell the printer to print the previously drawn points
c and lines which is drawn after the last 'stroke' is called

```



```

c
c *****
c      subroutine stroke
c      write(10,*) " stroke "
c      return
c      end
c *****
c
c      subroutine move
c
c          Move the current cursor position to (x,y)
c
c *****
c      subroutine move(x,y)
c      real x,y
c      real xscale, yscale, xtrans, ytrans
c      common /scale_vars/ xscale, yscale, xtrans, ytrans
c      write(10,*) (x+xtrans)*xscale , (y+ytrans)*yscale , " moveto"
c      return
c      end
c *****
c
c      subroutine rmove
c
c          Move the current cursor position by x and y along the
c      x and y direction respectively
c
c *****
c      subroutine rmove(x,y)
c      real x,y
c      real xscale, yscale, xtrans, ytrans
c      common /scale_vars/ xscale, yscale, xtrans, ytrans
c      write(10,*) x*xscale , y*yscale , " rmoveto"
c      return
c      end
c *****
c
c      subroutine draw
c
c          Draw a line from the current cursor position to (x,y)
c      and set the current cursor position to (x,y)
c
c *****
c      subroutine draw(x,y)
c      real x,y
c      real xscale, yscale, xtrans, ytrans
c      common /scale_vars/ xscale, yscale, xtrans, ytrans
c      write(10,*) (x+xtrans)*xscale , (y+ytrans)*yscale
c      & , " lineto"
c      return
c      end
c *****
c
c      subroutine clip
c
c *****
c      subroutine clip
c      write(10,*) " clip "
c      return
c      end
c *****
c
c      function name_req
c
c          Returns true if a is contained in b where a, b are
c      both character strings
c
c *****
c      function name_req(a,b)
c      character*(*) a,b
c      ia = len(a)
c      ib = len(b)

```

```

        name_req = .false.
        do 10 i=1,ib,ia
            if ( a .eq. b(i:i+ia-1) ) then
                name_req = .true.
                return
            endif
        10 continue
        return
    end

c *****
c
c function name_len
c
c     Returns the length of character string with the right
c     hand side spaces being trimmed
c *****
c
c     function name_len(a)
c     character*(*) a
c     do 10 i=len(a),1,-1
c         if ( a(i:i) .NE. ' ') then
c             name_len = i
c             return
c         endif
c     10 continue
c     name_len = 0
c     return
c     end

```

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. **ORIGINATOR** (the name and address of the organization preparing the document. Organizations for whom the document was prepared e.g. Establishment Sponsoring a contractor's report, or tasking agency, are entered in Section 8).

OPERATIONAL RESEARCH & ANALYSIS
DIRECTORATE OF AIR OPERATIONAL RESEARCH

2. **SECURITY CLASSIFICATION** (overall security classification of the document, including special warning terms if applicable)

UNCLASSIFIED

3. **TITLE** (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title)

THE DAOR SEARCH AND RESCUE (SAR) HELICOPTER TRANSIT MODEL

4. **AUTHORS** (last name, first name, middle initial)

Y.S. LEUNG AND G.L. CHRISTOPHER

5. **DATE OF PUBLICATION** (month Year of Publication of document)

NOVEMBER 1994

6a. **NO. OF PAGES** (total containing information. Include Annexes, Appendices, etc.)

75

6b. **NO. OF REFS** (total cited in document)

7. **DESCRIPTIVE NOTES** (the category of document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

DAOR RESEARCH NOTE 94/8

8. **SPONSORING ACTIVITY** (the name of the department project office or laboratory sponsoring the research and development. Include the address).

Directorate of Aerospace Requirements - Fighters and Transport
National Defence Headquarters
Ottawa, Ontario

9a. **PROJECT OR GRANT NO.** (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)

ORA Activity 23214 - Search and Rescue Operational Requirements
DAOR Project 23214-2 New SAR Helicopter Requirements and Analysis

9b. **CONTRACT NO.** (if appropriate, the applicable number under which the document was written.)

10a. **ORIGINATOR's document number** (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DAOR RESEARCH NOTE 94/8

10b. **OTHER DOCUMENT NOS.** (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. **DOCUMENT AVAILABILITY** (any limitations on further dissemination of the document, other than those imposed by security classification.)

(XX) Unlimited distribution

- () Distribution limited to defence departments and defence contractors: further distribution only as approved
() Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
() Distribution limited to government departments and agencies; further distribution only as approved
() Distribution limited to defence departments; further distribution only as approved
() Other (please specify):

12. **DOCUMENT ANNOUNCEMENT** (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNCLASSIFIED
SECURITY CLASSIFICATION OF FORM

13. **ABSTRACT** (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

The DAOR Search and Rescue (SAR) Helicopter Transit Model was developed to support the specification of minimum operating capabilities for a replacement SAR helicopter. For given set of helicopter characteristics and operating bases, the Model determines the optimal flight paths and transit times to travel to a distribution of available refuelling sites spread across the country. The Model was developed for the MULTICS computer system and used extensively to support the definition of the statement of requirements for a new SAR helicopter. After the statement of requirements for the SAR helicopter was approved, the Helicopter Transit Model was not used for some time. In that time the MULTICS system ceased operation.

In 1994, there was once again a requirement for the SAR Helicopter Transit Model. As the computer system for which the Model was designed was no longer operating, the Model has to be implemented on another computer system. The SAR Helicopter Transit Model was modified and implemented on a SUN Workstation. During the implementation process, the capabilities of the Model were enhanced.

This report documents the DAOR SAR Helicopter Transit Model and the work performed to return the Model to operational status. The report describes the procedures used by the Model to determine optimal flight paths and transit times. The report also provides a user guide to operate the Model.

14. **KEYWORDS, DESCRIPTORS or IDENTIFIERS** (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Optimization
Optimal Routing
Transit Routes
Helicopter Transit
Flight Paths
Transit Times
Area Coverage
Coverage Diagrams
Search and Rescue
SAR
SAR Coverage
SAR Travel Time

CanadaTM